UIUCDCS-R-77-869 UILU-ENG 77 1725

Inductive Inference in the Variable Valued Predicate Logic System VL_{21}: Methodology and Computer Implementation

by

James B. Larson

May 1977

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Inductive Inference in the Variable Valued Predicate Logic System VL_{21}: Methodology and Computer Implementation

by

James B. Larson

Department of Computer Science
University of Illinois
Urbana, Illinois

May 1977

This work was submitted in partial fulfillment of the requirements for the degree of Docotr of Philosophy in Computer Science in the Graduate College of the University of Illinois and was supported in part by the National Science Foundation, Grant No. NSF MCS 74-03514.

INDUCTIVE INFERENCE IN THE VARIABLE VALUED PREDICATE LOGIC SYSTEM VL_{21}: METHODOLOGY AND COMPUTER IMPLEMENTATION

James Burton Larson, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1977

A formal methodology and computer program are presented for the transformation of a set of user supplied logical decision rules into a new, generalized set of decision rules which is near optimal according to a user supplied criterion. The VL$2 logic system (a multi-valued version of a first order predicate calculus) is used as the framework for defining and expressing decision rules and transformations on decision rules. The program INDUCE_1 which implements certain inductive inference rules using a graphical representation of VL$2 expressions is described and some examples of inductive problems solved by the program are given.

ACKNOWLEDGEMENTS

I would like to express my gratitude to the following people for their help on this thesis: First, Professor R. S. Michalski for his inspiration, challenging problems, and many significant suggestions especially regarding meta-functions, my committee, Professor D. Plaisted and Dr. Don Friesen for encouragement and many helpful discussions. I especially wish to recognize the valuable support of Richard Chilausky during the early stages of this research, Barr Segal for proofreading of the manuscript, and the many predecessors in the area of inductive inference and variable valued logic whose efforts have been much appreciated in the development of this thesis. I am greateful for the financial support of the National Science Foundation, the Department of Computer Science, the Research Board of the University of Illinois, and the Computing Services Offices, the latter for supplying the computer time to implement the text formatter used for this paper.

Finally, without the moral encouragement of my wife, Rhonda, this work surely would have ended years ago from utter frustration and discouragement.

PREFACE

This paper was prepared on a CYBER 175 computer at the University of Illinois using a text formatter written by the author. Since several special characters were not available on the print train of the printer, the following character combinations have special meaning:

Character	Meaning
v	logical disjunction
&	logical conjunction
_&	set intersection
_E	the existential quantifier
_A	the universal quantifier

TABLE OF CONTENTS

1. Introduction

An important problem which is often presented to computer systems is that of extracting relevant information from complex data in order to gain a better understanding of the meaning behind such data. Most current methods are incapable of adequately describing highly structured situations and produce results which are difficult to interpret. A selection of those systems which overcome these difficulties is given in sections 1.2 and 1.3.

The following chapters deal with finding useful (as defined by the user with an optimality criterion), generalized information about sets of situations represented as logical VL decision rules. A decision rule is a form

CONDITION => DECISION

where CONDITION describes some set of situations and DECISION describes some new situation or action which is indicated if an existing situation satisfies the description in CONDITION. If no situation satisfies the CONDITION in a decision rule, the rule makes a NULL decision. The descriptions in CONDITION and DECISION are represented in the VL_2 logic system. This system is a variable valued first order predicate caluculus with a rich set of operators and

the facility for allowing user defined domain sizes and structures for variables and functions appropriate for the problem at hand. The approach taken here is to apply inductive inference rules to logical decision rules which express some decisions made with sets of situations in order to form new, near-optimal decision rules which retain the decision making capabilities of the original rules.

1.1 The Problem

The specific induction problem being investigated is as follows:

Given a set of decision rules:

$$
\begin{array}{llll}
C_{1,1} \Rightarrow D_1; & C_{1,2} \Rightarrow D_1; & \ldots & C_{1,t1} \Rightarrow D_1 \\
C_{2,1} \Rightarrow D_2; & C_{2,2} \Rightarrow D_2; & \ldots & C_{2,t2} \Rightarrow D_1 \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot \\
C_{n,1} \Rightarrow D_n; & C_{n,2} \Rightarrow D_n; & \ldots & C_{n,tn} \Rightarrow D_n
\end{array}
\qquad (1.1)
$$

where $C_{i,j}$ and D_i are expressions in the VL_2 system which represent the CONDITION and DECISION parts of decision rules respectively, then find, through an application of generalization rules, a set of VL decision rules:

$$C_1 \Rightarrow D_1$$
$$C_2 \Rightarrow D_2$$
$$\vdots$$
$$C_n \Rightarrow D_n \qquad (1.2)$$

which are with regard to the rules (1.1):

1) consistent
2) complete
3) optimal with regard to a user supplied optimality criteria.

The new rules are consistent if for any situation for which the new rules assign a decision (a non-NULL decision), the initial rules assign the same decision or a NULL decision. They are complete if for any situation for which the initial rules assign a decision, the new rules assign a decision. From the initial rules, it is usually possible to derive many sets of rules which are consistent and complete. Therefore, a criterion of optimality (defined by a user according to his problem) is used to select a few alternatives which are most

desirable according to the specific induction problem. The attention is restricted to sets of rules which make only one decision for a given situation.

1.2 Previous Specific Applications

Inductive inference is used here to describe a formal method for rewriting or generalizing available data in order to give new information about a problem and make new decisions which could not be obtained before. Statistical methods are probably the most widely used forms of inductive inference. These methods require a great deal of a-priori knowledge including the availability of a large set of data, knowledge about the interpendence of variables on each other, and an understanding of the type of underlying distribution of the data [Croft-71]. In addition, statistical results may be difficult to read and interpret [Larson 76] (e.g. a conditional probability matrix).

The first approach to automated inductive inference using logic was most likely developed by Hunt [Hunt 66]. He described a number of different schemes for generating decision trees which can be used to distinguish between sets of letter sequences. Although a decision tree produces an elegant procedure which can be easily executed on a computer, it lacks the flexibility necessary to represent more general concepts.

The HEURISTIC-DENDRAL program [Buchanan et.al. 69] provides a model appropriate for representing the structure of chemical compounds and some transformations representing possible chemical reactions which can be applied to the compound representations under certain known physical constraints. The program finds a set of possible structures of a compound knowing its empirical formula and mass spectrometer data by suggesting various structures for the compound and applying transformations to the structures under the guidence of a set of heuristics based on the mass spectrometer data. The meta-DENDRAL program [Buchanan et.al. 72] finds a general mechanism or theory which explains the transformations which take place relying on the knowledge of those transformations which are plausible and those which are forbidden.

Computer aided medical diagnosis is another area in which logical inductive inference methods have been suggested. Pople [Pople et.al. 72] have suggested a graph structure representation of biomedical facts and an approach to forming theories by finding common subgraphs using user supplied suggestions. Of particular note is work in the area of computer aided medical diagnosis and plant pathology [Michalski 73,74, Chilausky et.al 76, Larson 76] use the VL_1 system (a variable valued logic system which is the precursor

to the VL_2 system used here) and the program AQVAL/1-AQ7 to infer descriptions of clases of liver diseases and soybean diseases. The latter work is a specific application of a the VL_1 logic system to several problems.

Winston [Winston 70] demonstrates specific procedures which discover descriptions from examples in the toy blocks world. A description which is descriminant (i.e. can be used to distinquish one object or set of objects from other sets) is formed by matching the similar parts of the object under consideration with another object (near-miss) and then isolating the structures which are different between the two objects. This differs from the approach taken in the following chapters in that matching is only done here in order to adequately describe some specific feature which distinguishes between two objects. (e.g. to specify the second part from the top in an object, one may have to include some predicates which define the second from the top in terms of other descriptors. If the distinguishing feature involves this part, then the definition of the part used in the description must be common to both objects.) Winston also uses modifiers such as 'must', 'may', 'must not' in descriptions. A form of these modifiers is inherent in the VL_2 approach (e.g. discriminant descriptions involve the 'must' modifier, descriptive descriptions involving only set

of objects yields descriptions involving a type of 'may' modifier).

The program ARITHMETIC [Bongard 70] finds an algebraic rule which explains sample relationships. The program is given sets of tables, each table containing 3-tuples of a ternary relation. A set of 33 predicates are used in the program (although not explicitly given in the reference) where a predicate may be: e.g. if the quotient of the first two elements of the 3-tuple is positive, then the predicate is true. For each row of a table, a set of features is generated by finding Boolean combinations of the predicates applied to the row. A feature describing the table is generated for each Boolean combination by finding the product of Boolean combinations for all rows. The set of features which appear to be most useful in distinguishing one table from the others is selected as the description of each table.

1.3 Formal Systems for Inductive Inference

A criticism of some of the above endeavors is that they are problem specific. More general systems which have many possible application areas have emphasis in two types of approaches: 1) generation of descriptions of sets of objects represented in a logic system of some kind, and 2) creation

of new concepts in a sequential manner by generating and modifying hypotheses. A summary of several types of learning systems can be found in [Banerji 75]. Of particular interest here is the work of Morgan [Morgan 72] in which a formal system based on the first order predicate calculus with falsehood preserving transformations is presented. Briefly, the idea that inductive inference can be described as backwards reasoning is apparently not sufficent for a practical system. For example, if E_2 was derived from the assertions $\neg E_1$ and $E_1 \vee E_2$, then backward reasoning would somehow have to generate the two assertions above given only E_2. There are far to many expressions E_1 which can be applied to a situation such as this to yield a practical system. Instead, Morgan defines a falsehood preserving transformation of a deductive inference rule into an inductive inference rule (in Morgan's notation):

$$E_1 \mid\!-_F E_2$$

where E_2 is false for every interpretation in which E_1 is false. A set of transformations (F-rules) can be created to convert a deductive logic system into an inductive logic system (e.g. if E_1 and E_2 are atomic expressions, then

$$E_1 \vee E_2 \mid\!-_F E_1 E_2$$

i.e., from the disjunction $E_1 \vee E_2$, one may infer the conjunction $E_1 E_2$ while preserving falsehood). With this system, theorem proving techniques using falsehood preserving rules may be applied to inductive problems. (In later chapters, the symbol |< is used to denote such a 'generalizing' transformation.)

A number of authors have presented systems which use a graph structure representation of expression in a type of logic system. The 'parameterized structure representation' of Hayes-Roth [Hayes-Roth 76] is used in inductive tasks which learn descriptions of sets of objects and transformations from one set of objects to another set of objects from examples. The problems addressed are closely akin to the work in the following chapters of this paper, one of the objectives being to find the common properties of all examples of one class which are available to the system using a graphical representation. The number of possible alternatives in Hayes-Roth's method is limited by a fixed utility function which evaluates intermediate results and discards hypotheses of low utility. The work differs from that presented in the following chapters in that only independent descriptions are sought using a fixed utility criterion (these are called descriptive descriptions in

chapter 6.). The structure used for representation does not take into account specific domain structures which are inherent in the VL system and there is no facility for generating new descriptors within the system.

A formal approach using the predicate logic system [Vere 75] produces the largest common set of descriptors of a set of examples by representing examples using a graph structure and finding the largest common subgraph of these structures. Such methods suffer from the NP-complete nature of graph isomorphism algorithms (this problem is addressed in the following chapters by finding the smallest useful subgraphs of graphs of examples instead of the largest subgraph.). Neither Vere nor Hayes-Roth use negative examples heavily in their implementations.

Hedrick [Hedrick 74] uses a semantic net to represent examples and to build and modify hypotheses as new examples are given to the program. The semantic net supports only binary relations but otherwise looks similar to the graph structure of Vere and Hayes-Roth.

Kochen [Kochen 74] presents a different type of system with a set of initial events containing state variables, actions and relations between the actions, a learning program which applies certain transformations to events at various time steps. At each time step, weighted

hypotheses are formed which reduce the set of states stored in memory (i.e those states which are explained by the hypotheses).

Production systems provide a rich tool for the introduction of many inference techniques (recall that VL decision rules are similar to production rules). Briefly, a production system architecture contains a working memory, a set of productions which modify working memory, and a recognize-act cycle with conflict resolution to dictate the order in which productions are applied to memory and to add new productions when necessary.

Waterman [Waterman 70, 74, 75] uses these to solve several problems. A program which plays poker has been developed [Waterman 70] which designs betting strategies in terms of production rules. More recently [Waterman 75], the approach has been applied to recognizing letter sequences with success. Rychener [Rychener 76] has applied production systems to chess end games and natural language input of a toy blocks world. These two authors use distinct production system architectures which differ in the ordering of working memory, ordering of productions and in the way in which new productions are added to an existing set of productions to correct errors made by the system.

As a final note with regard to production systems,

the MYCIN system [Shortlieffe 74] has been shown to be a very powerful tool in aiding physicians with regard to antimicrobial therapy selection. In the MYCIN system, deductive inference using a multi-valued truth model is applied to expert supplied productions and a data base consisting of a patient record.

1.4 Overview of the Following Chapters

The VL_{21} system (a subset of the VL_2 system) is described in chapter 2. The subset used here uses only the truth values TRUE, FALSE, and UNKNOWN instead of the multi-valued truth domain of the VL_2 system. Also, only a very small subset of the operators available in VL_2 are used. The inductive rules used to transform initial decision rules (1.1) into new decision rules (1.2) are given in chapter 3. The rules involve selecting the most significant features of the initial condition ($C_{i,j}$ in 1.1), extending the value set which each feature may assume under the domain structure constraints, and adding new global functions which describe certain characteristics of the condition ($C_{i,j}$). Chapter 4 contains a graphical representation of VL_2 rules. A subset of the graph structure is used in the computer program INDUCE_1 described in chapter 5. The program accepts as input:

1) a set of decision rules representing certain examples of sets of decisions,
2) a problem environment description in the form of VL decision rules which describes certain characteristics of functions and domains which arise from the particular application, and
3) a set of control parameters which supply the optimality criterion and certain parameters which limit the number of alternatives generated at various points in the program.

The output from the program contains a set of complete, consistent decision rules. Chapter 6 gives results of the program as applied to some specific situations. Chapter 7 describes some limitations and possible extensions of the program. Two appendices are given: appendix A, providing a listing of the detailed output of the program applied to one example of chapter 6, and appendix B, giving a brief review of the precursor to this work (the program AQVAL/1-AQ7) which is used as a procedure in INDUCE_1.

2. Representing Decisions in the VL_2 System

Much of the information in this chapter is found in [Larson et.al. 77, Michalski 74b]. It is included here to give the reader a familiarity with the VL_2 system. The complete VL_2 system contains a very rich set of operators and domains. A subset of VL_2 called VL_{21} which contains a basic set of operators and domains is used here. Only the VL_{21} system is described with notes indicating the extensions which are possible in the full VL_2 system. In later chapters, the notation VL_2 is used to refer to the system VL_{21}.

The logic system VL_{21} is a language for describing situations (e.g. objects, classes of objects) and expressing decision and inference rules. The language provides for a compact expression of descriptions which is both easily readable and sufficiently precise to facilitate formal manipulation (possibly by a computer).

There are two major differences between VL_{21} and the first order predicate calculus

1. Instead of predicates, selectors are used which can be viewed as tests for membership of values of predicates and functions in a certain set.

2. Each variable, predicate and function symbol is assigned a domain (or value set) together with a characterization of the structure of the domain. (This feature facilitates the process of rule generalization and allows for the application of different generalization transformations according to the structure of the domain.)

There are three types of domains currently distinguished:

1. Unordered or Nominal

   Elements of the domain are considered to be independent entities; no structure is assumed to relate them. A variable or function symbol with this domain is called nominal or cartesian (e.g. blood type, names of objects, etc.).

2. Linearly Ordered or Interval

   The domain is a linearly ordered set. A variable or function symbol with this domain is called interval (e.g. military rank, temperature, size).

3. Tree Ordered

Elements of the domain are ordered into a tree structure. A predecessor node in the tree represents a concept which is more general than the concepts represented by the descendent nodes (e.g., the predecessor of the nodes 'triangle', 'rectangle', 'pentagon' may be a 'polygon'). A variable or function symbol with such a domain is called structured.

2.1 VL System Structure

The VL_{21} system used is a 5-tuple (V,F,S,R,I) where:

V - is a set of variable symbols. Each variable symbol is associated with a domain $D(x_i)$. A group of variables which have the same domain are labelled with the same variable symbol but a different subscript (e.g. $x_1,x_2,\ldots,x_k$, $y_1,y_2,\ldots,y_l$ are specifications of variables in two variable groups which assume values from two domains denoted D(x) and D(y) or alternatively, $D(x_i)$ and $D(y_i)$.

F - is a set of n-ary functions and predicate symbols. Each n-ary function symbol represents a mapping from an argument space into a domain. For a function $f(x_1,x_2,\ldots,x_n)$, this is a mapping:

$$D(x_1) \times D(x_2) \times \ldots \times D(x_k) \rightarrow D(f)$$

where $D(x_1)$, $D(x_2)$, ... $D(x_k)$, $D(f)$ represent the domains of the variables x_1, x_2, ... x_k and the domain of the function f, respectively. A predicate is a function whose domain is the set [TRUE,FALSE]. Included in the domains of all function and variable symbols is the value NA (not applicable).

s - is a set of symbols including:

( ) [ ] = < > -> <-> v => & v E E. A A. , .

R - is a set of formation rules described in section 2.3

I - is a set of interpretation rules described in section 2.4

2.2 Selector Formation and Interpretation Rules

A well formed VL_2 formula (wff) is composed of quantifier forms, selectors, and logical connective symbols.

A selector is a form:

[L # R] or [L']

where

L, L' - each called the referee are atomic forms. An atomic form is a variable symbol or a function or predicate symbol followed optionally by a list of atomic forms enclosed in parentheses. In the above forms, the atomic form L' must be a predicate symbol with arguments following in parentheses. If L contains a function symbol, then the related function is called an atomic function.

R - the reference is a set of values in the domain of the atomic function of L. R may be in several forms:

Reference Example	Description
a	a constant in the domain of the atomic function of L
a,b	a list of values in the domain of L separated by commas
a..b	a pair of values in the domain of L separated by (..)
*	the symbol (*) representing

	all values in the domain of L (except NA)
NA	the value NA (not applicable)

\# - is one of the symbol combinations

= <= >= < > ¬=

If R is a set of values, then L is related to R by # if

when # is = or ¬=	L has a value (does not have a value) in the set R
when # is <= >= < >	L has a value related to every value of R by #.

The selector is interpreted as a unit of information about a situation with value or truth-status TRUE if the relation R # L holds or FALSE if the relation does not hold, or UNKNOWN in which case the selector is interpreted as a question about the situation which must be answered in order to determine if the selector is satisfied. If some

variables in the atomic form of the selector are quantified, these quantifiers must be considered when determining the truth-status of a selector.

If R is *, then L is related to R for any value of L except NA (in this case, * is always =). Below are some examples of a selector:

Selector	Interpretation: truth-status TRUE
[color(wall$_1$) = white]	The color of the wall represented by wall$_1$ is white.
[length(box$_1$) $\geq$1]	The length of the box represented by box$_1$ is greater than or equal to 1.
[box$_1$ = 2..5]	The variable box$_1$ may have a value between 2 and 5 inclusive. The values of box$_1$ may represent various boxes in a situation. The selector restricts the range of values of the variable box$_1$ to the values 2 through 5.
[ontop(x_1,x_2)]	The part represented by x_1 is on top of the part represented by x_2.

2.3 VL Formation Rule

Formulas in the VL_2 logic system are used to describe situations, and also to express decision rules and inference rules. The VL formulas are defined by the following formation rules:

1. A selector is a VL formula (wff).

2. If V, V_1 and V_2 are wff, then so are:

(V)	a formula in parentheses
¬V	inverse
V_1 & V_2 or V_1V_2	conjunction (the symbol & is used to represent conjunction)
V_1 v V_2	disjunction
V_1 $\underline{v}$ V_2	exclusive disjunction
V_1 ⟍⟋ V_2	exception
V_1 -> V_2	V_1 implies V_2
V_1 <-> V_2	V_1 is equivalent to V_2

$\underline{E}x_1,x_2,\ldots,x_k(V)$ Existentially quantified formula ($\underline{E}$ is used to represent the existential quantifier).

$\underline{E}.x_1,x_2,\ldots,x_k(v)$ Distinctly existentially quantified formula

$\underline{A}x_1,x_2,\ldots,x_k(v)$ Universally quantified formula ($\underline{A}$ is used to represent the universal quantifier).

Not all of these forms are considered in the following chapters. In chapter 3, (VL inference rules) only conjunction, disjunction, and quantifiers are considered. Chapter 4 (Graph Representation) presents a graph structure representation which includes all of these forms but the types of formulas actually included in the algorithm and the implementation involve only conjunction and distinct existential quantification.

2.4 Interpretation Rules

A VL formula may have truth-status TRUE, FALSE, or UNKNOWN. In the full VL_2 system, a truth-status domain with interval structure may be defined, but here only the values

above are considered. The connectives ($\neg$ v &) are interpreted in the normal manner:

VL formula	Interpretation
$\neg V$	FALSE if V is TRUE, TRUE if V is FALSE, UNKNOWN if V is UNKNOWN.
V_1 v V_2	TRUE IF EITHER V_1 OR V_2 IS TRUE, UNKNOWN if both V_1 and V_2 are UNKNOWN or one is UNKNOWN and the other FALSE, TRUE otherwise.
V_1 & V_2	UNKNOWN if both V_1 and V_2 are UNKNOWN or one is true and the other is UNKNOWN, TRUE if both V_1 and V_2 are TRUE, FALSE otherwise.

The remaining connectives may be rewritten in equivalent forms:

VL Formula	Equivalent form
$V_1 \rightarrow V_2$	$\neg V_1 \vee V_2$
$V_1 \leftrightarrow V_2$	$(V_1 \rightarrow V_2) \,\&\, (V_2 \rightarrow V_1)$
$V_1 \diagdown V_2$	$V_1 \neg V_2$
$V_1 \underline{\vee} V_2$	$V_1 \vee V_2 \diagdown V_1 V_2$

A VL system is used to describe a set of situations. In order to effectively apply a formula to a set of situations, the VL_2 system should contain variables, functions, and predicates which adequately characterize the situations. To determine the truth-status of a formula with regard to a specific situation, an event is created (an event may be viewed as an interpretation of a situation in the VL_2 system). An event is a sequence of assignments to variables, functions and predicates in the system which characterize a specific situation. Quantified variables may be assigned a set of values. One function assignment may be made to a given set of values of arguments if the value of the function is known. If a function does not have an assignment for a given set of values, then the value NA (not applicable) is assumed.

A selector [L # R] (or [L']) is satisfied by an event if there is a set of assignments to variables and functions (or predicates) in L (or L') such that L is related to R by # (or L' has the value TRUE).

A VL_2 formula is satisfied by an event if it has truth status TRUE when applied to the event.

The quantified formulas are interpreted:

The truth status of

$\underline{E}x_1,x_2,\ldots,x_n$ (V) is TRUE (or FALSE) in a given situation if there exists (or does not exist) values for $x_1,x_2,\ldots,x_n$ in the event assignments which makes the truth-status of the formula V equal to TRUE

? - if it is not known whether there exist values ...

$\underline{E}.x_1,x_2,\ldots,x_n$ (V) is TRUE (or FALSE) in a given situation if there exists (or does not exist) <u>distinct</u> (different) values for $x_1,x_2,\ldots,x_n$ in the event

assignments which makes the truth-status of the formula equal to TRUE. This obviates the need for extra predicates in an expression like $x_1 \neg= x_2$, $x_2 \neg= x_3$, $x_1 \neg= x_3$, etc.

? - if it is not known whether there exist values ...

$\underline{A}x_1, x_2, \ldots, x_n$ (V) is TRUE (or FALSE) in a given situation if for all assignments to the varialles $x_1, x_2, \ldots, x_n$, the formula V has truth-status equal to TRUE.

A VL_2 formula is a <u>description</u> of a situation if every event which can be derived from the situation satisfies the VL_2 formula and every event which satisfies the formula is also an interpretation of the situation.

2.5 VL Decision Rules

If V_1 and V_2 are VL_2 formulas, a general form of a VL decision rule is

$$V_1 \Rightarrow V_2 \qquad (2.5.1)$$

The formula V_1 is called the condition part and V_2 is the decision part. A restricted form of the VL_2 decision rule will be used in the following chapters.

(In the computer implementation, the formula V_2 is assumed to be a product of selectors which contain 0-ary functions in the referee. The terminology is relaxed in this case to allow the function symbol which appears in the decision part to be called a decision variable.)

A decision rule in the form 2.5.1 may be applied to a set of situations as follows: If the condition part of the decision rule (V_1) is given truth-status TRUE, then the decision part of the decision rule (V_2) also assumes the truth-status TRUE. For each event (assignment e:=L) which satisfies V_1, a new set of assignments are made to the event using the decision part of the rule to form the set of all events which satifsfy the conjunction V_1 & V_2. For example, given the decision rule:

$$\underline{E}.x_1,x_2\ [\,p(x_1,x_2)\,] \Rightarrow [D=1] \qquad (2.5.2)$$

and the event

$$e: \quad x_1:=0,1;\ \ x_2:=0,1\ \ p(0,1):=TRUE \qquad (2.5.3)$$

with variables functions and predicates: x, D, p, q with

domains D(x)=[0,1], D(D)=[0,1], D(p),D(q)=[TRUE,FALSE], a new assignment is made to e to give:

$$e: \quad x_1:=0,1; \quad x_2:=0,1; \quad p(0,1):=TRUE; \quad D:=1. \qquad (2.5.4)$$

A decision rule:

$$\underline{E}.x_1,x_2\,[p(x_1,x_2)] \Rightarrow \underline{E}.x_1,x_2\,[q(x_1,x_2)] \qquad (2.5.5)$$

applied to the event 2.5.3 gives one of the two new events:

$$e_1: \quad x_1:=0,1; \quad x_2:=0,1; \quad p(0,1):=TRUE; \quad q(0,1):=TRUE;$$
$$e_2: \quad x_1:=0,1; \quad x_2:=0,1; \quad p(0,1):=TRUE; \quad q(1,0):=TRUE;$$

Note that q(1,1) and q(0,0) are not given status TRUE since the quantifier $\underline{E}.$ insists that the two variables x_1 and x_2 have different values.

Given a set of decision rules each in the form 2.5.1, the set may be applied to a set of events. Initially, the condition parts of all decision rules have value UNKNOWN. If an event satisfies the condition part of a decision rule, new assignments are made to the event according to the decision part of the satisfied rule and the condition part of the rule returns to truth status UNKNOWN.

In the remaining chapters, events are only used as a formal basis for defining certain concepts. Since the number

of events necessary to completely describe a situation is quite large, only the VL_2 formulas themselves are manipulated by the algorithms.

3. VL Transformation Rules

From one set of decision rules (1.1), a new set of decision rules (1.2) is obtained by applying certain transformation rules (t-rules). For now, we will restrict our attention to rules which transform the condition part of a rule. These t-rules may be grouped into three types of rules based on the events which satisfy the condition part.

Given two rules:

$$R_1 : V_1 \Rightarrow D$$
$$R_2 : V_2 \Rightarrow D$$

V_1 is <u>more general</u> than V_2 if every event satisfying V_2 also satisfies V_1. If the converse is also true, then V_1 is equivalent to V_2. Below, R_1 and R_2 are the <u>input</u> and <u>output</u> of a t-rule. The three types of t-rules can be expressed as follows:

A transformation T : $R_1 \mid\# R_2$ (# being one of =, <, >) is

1. An equivalence transformation (denoted $R_1 \mid= R_2$) if the condition parts of both rules are equivalent.

2. A generalizing transformation (denoted $R_1 \mid< R_2$) if

the condition part V_2 of R_2 is more general than the condition part V_1 of R_1.

Rules 1. and 2. are called <u>inductive inference rules</u>

3. A specializing transformation (deductive inference rule denoted R_1 |> R_2) if the condition part V_1 is more general than the condition part V_2.

Here, we are most interested in the first two types of t-rules (i.e., inductive inference rules).

3.1 Equivalence Transformation

An equivalence transformation rewrites a VL_2 formula into a different form either using equivalent VL_2 operators or introducing new functions which represent some information already in the rule in a different manner. Below are some examples of equivalence transformations. The symbols L_1 and L_2 represent atomic forms, V and V' represent VL_2 formulas, D represents a VL_2 formula which has no variables in common with V, V', L_1 or L_2, and |= is used to represent an equivalence transformation.

E1. Equivalent VL_2 forms.

V([L = 1] v [L = 2]) => D |= V[L = 1,2] => D

V[L ¬= 3,4] => D |= V[L = 0,1,2,5] => D

(assuming that the domain of L is the interval [0..5] and has nominal structure). The dot operator (.) between two atomic forms is called 'internal conjunction' thus the expression on the right above is read: 'If L_1 and L_2 both have the value 1 and V is satisfied, then make decision D.'.

V([L_1 = 1][L_2 = 1]) => D |= V([L_1.L_2 = 1]) => D

(assuming that L_1 and L_2 have the same domain size and structure).

E2. Internal Conjunction of Arguments

VV' => D |= VV'[f'(x_1.x_2)=i] => D

V' = [f(x_1) = i][f(x_2) = i]

This rule introduces a new predicate f' which has the domain [TRUE,FALSE] and two arguments. The (.) operator instead of (,) indicates that the order of arguments to f' is irrelevant. The function f' assumes the value TRUE if f has the value i for both arguments x_1 and x_2.

E3. Introducing New Predicates.

VV' => D |= VV'[rel_f(x_1,x_2)] => D

$V' = [f(x_1) = i][f(x_2) = j]$

where i is related to j by rel. For example, i<j, i>j, i>=j would result in new predicates LT_f, GT_f, GE_f.

E4. Splitting the Condition Part.

$V \vee V' \Rightarrow D \quad \models V \Rightarrow D,\ V' \Rightarrow D$

This rule is used to form a set of decision rules with product condition parts from decision rules in disjunctive normal form.

3.2 Generalizing Transformations

This type of transformation usually produces not only a more general decision rule from a set of decision rules but also a 'simpler' one than the original. Some rules are applied in context. That is, one rule is actually transformed but the context consisting of rules with a related decision is used to obtain a more optimal result. (These transformations may also be interpreted as transformations from a set of rules into a new, more general rule. The approach of focussing on one rule in the 'context' of others is taken here to more closely reflect the approach taken in the implementation.) In the following rules, the symbol |< is used to indicate an inductive inference rule.

G1 Dropping a Selector.

V[L = R] => D |< V => D

Although this rule is interesting in a formal sense, it should be applied with care since the number of generalizations possible with successive applications of this rule is very large. More is said about this problem in section 3.4.

G2. Extending the Reference

Nominal domain structure

V[L = a] => D |< V[L = a,b] => D

in context [L = b] => D

Interval domain structure

V[L = a] => D |< V[L = a..b] => D

in context [L = b] => D

Tree structured domain

V[L = a] => D |< V[L = c] => D

in context [L = b] => D

c is a predecessor of both a and b in the generalization structure of the domain of L.

G3. Extension Against

Nominal domain

$V_1[L = R_1] \Rightarrow D \quad |< \quad V_1[L \neg= R_2] \Rightarrow D$

in context: $V_2[L = R_2] \Rightarrow \neg D$

assuming $R_1 \underline{\&} R_2 =$ null. (The symbol $\underline{\&}$ denotes set intersection.)

Interval domain structure

$V_1[L = a..b] \Rightarrow D \quad |< \quad V_1[L = e..f] \Rightarrow D$

in context: $V_2[L = c..d] \Rightarrow \neg D$

assuming $[a..b] \underline{\&} [c..d]$ = null

if $b < c$ then $e = 0$, $f = c-1$

if $a > d$ then $e = d+1$, $f = \underline{h}$ (0 and $\underline{h}$ are the minimum and maximum elements in the domain of L)

Tree structured domain

$V_1[L = a] \Rightarrow D \quad |< \quad V_1[L = c] \Rightarrow D$

in context: $V_2[L = b] \Rightarrow \neg D$

assuming $a \underline{\&} b$ = null

the constant (c) is the most distant ancestor of (a) which is not an ancestor of (b) (c may be equal to a).

G4. Replace $\underline{A}$ with a constant

$(\underline{A}x_i\ V(x_i)) \Rightarrow D \quad |< \quad V(x_i)[x_i = a] \Rightarrow D$

a - an element in the domain of x_i. (The symbol $\underline{A}$ represents the universal quantifier.) For example:

$\underline{A}x_1\ [f(x_1) = 1] \Rightarrow D \quad |< \quad [f(x_1) = 1][x_1 = 2] \Rightarrow D$

The left side of the expression requires that f have the value 1 for all values of x_1 before a decision be made. The generalized expression (right side) only requires examination of one value in the domain of x_1, namely the value 2. All other values in the domain of x_1 are irrelevant to the decision rule.

G5. Replace a constant with $\underline{E}$

$V(x_i)[x_i = a] \Rightarrow D \quad |< \quad \underline{E}x_i\ V(x_i) \Rightarrow D$

a - an element in the domain of x_i. (The symbol $\underline{E}$ represents the existential quantifier.) For example:

$[f(x_1) = 1][x_1 = 3] \quad |< \quad \underline{E}x_1\ [f(x_1) = 1] \Rightarrow D$

The left side of the expression makes a decision only if f has the value 1 for x_1 with value 3. The

generalized expression makes a decision if f has the value 3 or any value of x_1.

G6. Move $\underline{E}$ to the Right

$$\underline{E}x_i\ \underline{A}x_j\ V(x_i,x_j) \Rightarrow D \quad |< \quad \underline{A}x_j\ \underline{E}x_i\ V(x_i,x_j) \Rightarrow D$$

3.3 Specializing Transformations

A specializing transformation may be used to apply a decision rule to a new situation or to add certain restrictions to decision rules. Below, the symbol |> represents a deductive inference rule.

R1. Adding Restrictions

$VV' \Rightarrow D \quad |> \quad VV'V'' \Rightarrow D$

in context $V' \Rightarrow V''$

This rule is used to add restrictions to descriptions and generalizations where the restrictions represent some structure which is imposed on a function or some relationship between functions which always holds. For example, the transitivity or symmetry of a function may be introduced using a restriction.

$[f(x_1,x_2)][f(x_2,x_3)] \Rightarrow [f(x_1,x_3)]$ (transitivity)

$[f(x_1,x_2)] \Rightarrow [f(x_2,x_1)]$ (symmetry)

R2. Dropping Product Rule

$V_1 \wedge V_2 \Rightarrow D \quad |\!> \quad V_1 \Rightarrow D$

This rule may be used to obtain a set of decision rules with a product in the condition part from a rule in disjunctive normal form.

3.4 Transformation Rules Involving the Decision Part

It is clear that reversing the roles of the input and output of an inductive t-rule gives a deductive t-rule and conversely. Therefore, each of the rules G1-G6 could be inverted to get a deductive rule. The rules in sections 3.1-3.3 were based on the events satisfying the condition part of a decision rule. Similar rules may be applied to the decision part of a rule to obtain equivalent, more general or more restricted rules.

Given two rules:

$R_1 : V \Rightarrow D_1$

$R_2 : V \Rightarrow D_2$

R_2 is more general than R_1 and R_1 is more restricted than R_2 if every assignment of event values made by R_1 is also made

by R_2. If the converse is also true, then R_1 is equivalent to R_2. To transform the decision part of a decision rule, apply G1-G6 to the decision part of a rule to obtain a deductive rule and R1-R2 to obtain an inductive rule interchanging the roles of the condition and decision parts in the transformation rules. A few examples are given below.

DP1. Dropping Selector Rule Applied to the Decision Part

$$V \Rightarrow [L_1 = R_1][L_2 = R_2] \quad |> \quad V \Rightarrow [L_1 = R_1]$$

Though this is a generalization rule when applied to the condition part, it is a deductive rule when applied to the decision part of a decision rule.

DP2. Splitting the Decision Part

$$V \Rightarrow [L_2 = R_2]D \quad |= V \Rightarrow [L_1 = R_1],\ V \Rightarrow D$$

This equivalence preserving rule is used to produce a set of decision rules which involve only one decision variable.

DP3. Replace a Constant with $\underline{A}$ in the Decision Part

$$V \Rightarrow [L(x_1) = R_1][x_1 = a] \quad |> \quad V \Rightarrow \underline{A}x_1\,[L(x_1) = R_1]$$

3.5 Example of Application of Transformation Rules

In this section, selected transformations will be examined in more detail with respect to a specific example. Consider the situation in Figure 3.1. There are four objects classified according to two decision variables DA and DB. Each object is described in terms of the following VL_2 functions and predicates:

P_1, P_2: variables each representing a part in an object with domain : P = [0,1] with nominal structure

ontop: a predicate mapping P x P into [TRUE,FALSE]

The predicate is TRUE if the part represented by the first argument is on top of the part represented by the second argument.

shape: a function mapping P into [Triangle(T), Circle(C), Rectangle(R), Ellipse(E), Polygon(P), Curved Figure(CF)] with generalizations: [shape=T,R]=>[shape=P] and [shape=C,E]=>[shape=CF].

The function specifies the shape of the part represented by the argument. The domain is a tree structured domain where Polygon is an ancestor (a generalization) of Triangle and Rectangle and Curved Figure is an ancestor of Circle and Ellipse.

DA,DB: decision variables with domain [1,2]

	DA = 1		DA = 2	
1	Circle Triangle	2	Circle Circle	DB = 1
3	Ellipse Rectangle	4	Rectangle Rectangle	DB = 2

Figure 3.1 Set of Objects

Each object in fig 3.1 may be described in the VL_2 system. The description of objects 1 and 4 are:

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=C][\text{shape}(p_2)=T] \Rightarrow [DA=1][DB=1] \quad (3.5.1)$$

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=R][\text{shape}(p_2)=R] \Rightarrow [DA=2][DB=2] \quad (3.5.2)$$

The variables p_1 and p_2 are assumed to be quantified with the operator E. (distinct existential quantifier). Using the

rule DP1, the decision rule 3.5.1 can be transformed into two new rules:

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=C][\text{shape}(p_2)=T] \Rightarrow [DA=1] \quad (3.5.3)$$

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1]=C][\text{shape}(p_2)=T] \Rightarrow [DB=1] \quad (3.5.4)$$

Concentrating now on 3.5.4, the dropping selector rule may be applied twice in succession to obtain

$$[\text{shape}(p_1)=C] \Rightarrow [DB=1]. \quad (3.5.5)$$

This rule describes all objects in figure 3.1 with the decision DB=1. In addition, it does not describe any object with DB=2; so it is a complete, consistent generalization of the objects with DB=1. (Note that completeness and consistency are in no way guaranteed by the application of the dropping selector rule. These conditions can however be checked after each application of the rule.) Applying the extension against rule to 3.5.3 in the context

$$[\text{onto}(p_1,p_2)][\text{shape}(p_2)=R][\text{shape}(p_2)=R] \Rightarrow [DA=2] \quad (3.5.6)$$

focusing on the selectors $[\text{shape}(p_1)=C]$ in 3.5.4 and $[\text{shape}(p_1)=R]$ in 3.5.6 one may obtain

$$[\text{shape}(p_1=CF][\text{ontop}(p_1,p_2)][\text{shape}(p_2)=T] \Rightarrow [DA=1]. \quad (3.5.7)$$

Applying extension against now to 3.5.7 in the context

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=C][\text{shape}(p_2)=C] \Rightarrow [DA=2] \quad (3.5.8)$$

focusing on the selectors $[\text{shape}(p_2)=T]$ in 3.5.7 and $[\text{shape}(p_1)=C]$ in 3.5.8, one obtains

$$[\text{shape}(p_1)=CF][\text{ontop}(p_1,p_2)][\text{shape}(p_2)=P] \Rightarrow [DA=1]. \quad (3.5.9)$$

This rule is a consistent and complete generalization of all rules in fugure 3.1 with DA=1 in the decision part.

Looking now at the description of object 4 (3.5.2), two rules are obtained by applying t-rule R1:

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=R][\text{shape}(p_2)=R] \Rightarrow [DA=2] \quad (3.5.10)$$

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=R][\text{shape}(p_2)=R] \Rightarrow [DB=2] \quad (3.5.11)$$

An application of the dropping selector rule to 3.5.11 twice in succession gives a complete, consistent rule:

$$[\text{shape}(p_1)=R] \Rightarrow [DB=2] \quad (3.5.12)$$

An application of t-rule E3 to 3.5.10 with a relation <u>equals</u> produces:

$$[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=R][\text{shape}(p_2)=R][\text{EQ-shape}(p_1,p_2)]$$

$$\Rightarrow [DA=2] \qquad (3.5.13)$$

The predicate EQ-shape specifies that the value of the function shape is the same for arguments p_1 and p_2. The (.) separating the arguments p_1 and p_2 of EQ-shape signify that the order of the arguments is irrelevant. (In the implementation, a selector with this type of predicate is written [shape(p_1.p_2)=same].) An application of the dropping selector rule three times in succession to 3.5.13 gives:

$$[\text{EQ-shape}(p_1,p_2)] \Rightarrow [DA=2] \qquad (3.5.14)$$

which is a complete, consistent generalization of 3.5.10.

In summary, the new rules which were obtained are:

$$\begin{aligned}
&[\text{ontop}(p_1,p_2)][\text{shape}(p_1)=CF][\text{shape}(p_2)=P] \Rightarrow [DA=1]\\
&[\text{EQ-shape}(p_1,p_2)] \Rightarrow [DA=2] \qquad (3.5.15)\\
&[\text{shape}(p_1)=C] \Rightarrow [DB=1]\\
&[\text{shape}(p_1)=R] \Rightarrow [DB=2]
\end{aligned}$$

In the above discussion, the resulting simple generalizations depended on knowing the proper rules to apply and the proper portions of the decision rules to modify. For larger problems, this approach is infeasible because of the large number of possible generalizations which could be

made. Therefore, a more efficient approach is required. Such an approach is given in the next section and described in detail in the chapter on basic procedures (chapter 5).

3.6 Efficient Application of Generalization Rules

There are two significant problems with using the procedure described in section 3.5 to apply t-rules to a set of decision rules. The first is the large number of new rules which can be generated from one rule. This problem could be circumvented by trimming the intermediate lists of new rules selecting only the most promising set of rules according to a user specified criterion before further application of t-rules (see section 5.3 for a description of a trimming procedure). A second problem which is not surmounted as easily is the complexity involved in determining whether a VL_2 formula is consistent and in evaluating the optimality cost functions during trimming. To determine whether one VL_2 formula is a generalization of another or is consistent with respect to another formula, one must determine whether all or any of the events which satisfy one VL_2 formula also satisfy another VL_2 formula.

Using the current implementation, this involves determining whether one graph structure representation is a subgraph of another (see section 5.4 for a description of a

subgraph isomorphism algorithm). This problem is exponential in nature (i.e., the time to determine whether one graph is a subgraph of another using a depth-first-search is proportional to m raised to the power of n where m is the number of nodes in the larger graph and n is the number of nodes in the smaller graph). Actually, it is not quite so time consuming since the graphs have relatively few edges and the edges and nodes are labelled. It is, however, important to form simple generalizations which correspond to small graph structures.

Since the cost criterion normally includes a cost function which minimizes the number of selectors in a product, the consistency and optimality of optimal rules should be easily calculated. In the example of section 3.5, the original rules were made smaller by dropping selectors. An alternate approach is to grow the generalized rule beginning with single selectors and adding new selectors until a consistent rule or set of alternative rules is created. A very general algorithm follows; chapter 5 gives a complete description of the algorithm in the current implementation.

1. Given a set of decision rules in disjunctive normal form, create a new set of rules with a product in the

condition part and one selector in the decision part of each rule.

2. Select a rule which involves one value of a decision variable. Add to the rule new selectors which represent functional relations as specified by a user. (i.e., multiply the condition part of the rule by selectors containing new functional relations.)

3. Find a consistent generalization of the rule from 2 by locating the most promising selectors of the rule from 2 and adding new selectors to each of these selectors until a set of consistent generalizations of the rule from 2 is obtained.

4. Apply the extension against rule using an AQVAL/1 procedure to all consistent rules.

5. Select the best generalization from this set and remove rules from the set produced in step 1 for which this is a generalization. Repeat steps 2-5 until no more rules remain which involve the decision variable and value of step 2.

6. Continue by selecting another value of the current

decision variable or selecting another decision variable until all decisions have been considered.

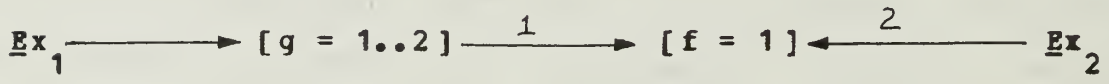
4. Computer Representation of VL Decision Rules

Some of the information in this section appears in [Larson et.al. 77] and is given here as a background for the description of the computer implementation. A VL decision rule can be represented as a graph with labelled nodes and directed labelled edges. The labels on the nodes can be: a) a selector containing k-ary descriptors without argument lists, b) a k-ary descriptor without arguments, c) a quantified variable with an optional subrange of values, d) a logical operator. (From here on, a node is referred to by its label, e.g., a selector node means a node with a selector label.) The edges are labelled with integers from 0,1,... . Edges not labelled 0 refer to the position of an argument in the label at the head of the edge. (Edges have non-zero labels only if the position in the argument list of the head node is important. Labels of 0 may be dropped for convenience.)

Several different types of relations may be represented by edges. The type of relation is determined by the label on the node at each end of the edge. The types of relations are:

1. Function Dependence - The label of the head node of

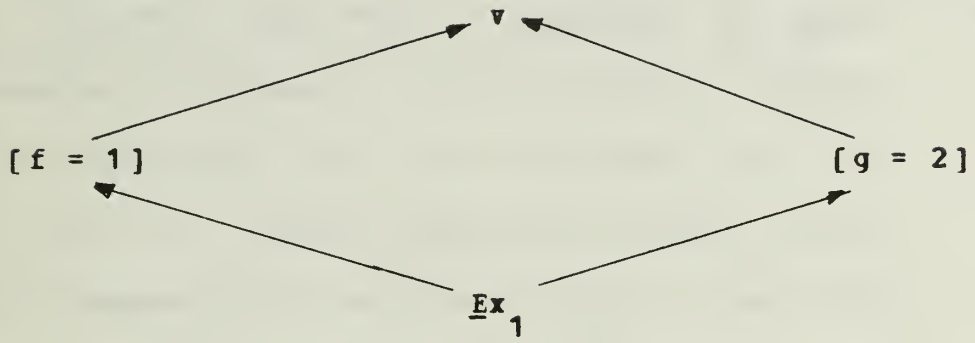
the edge has a k-ary descriptor. The value represented by the edge is the value of the atomic form in the tail if the tail is a selector node, a descriptor value if the tail is a descriptor node, or one or all of a set of descriptor values if the tail is a quantified variable. The edge label specifies which argument of the head node assumes this value (Figure 4.1).



Functional Dependence: $\underline{E}x_1,x_2([g(x_1)=1..2][f(g(x_1),x_2) = 1]$

Figure 4.1

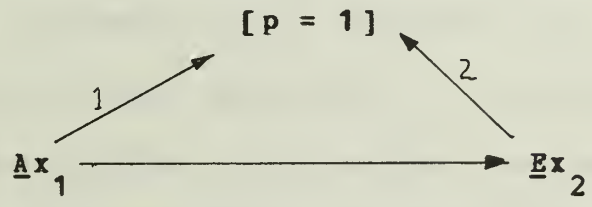
2. Logical Dependence - The head node is a logical operator (e.g. v, &, =>) and the tail node is a selector node, or a logical operator node. If the tail node is a selector, then the value represented by the edge is the truth value of the selector at the tail (Figure 4.2)



Logical Dependence : $\underline{E}x_1([f(x_1) = 1] \vee [g(x_1) = 2]$

Figure 4.2

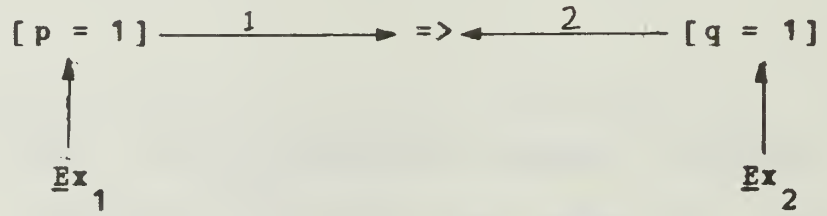
3. Implicit Variable Dependence - The labels of the head and tail nodes are quantified variables. This type of dependence represents the implicit function (which can be represented by a Skolem function) (Figure 4.3).



Implicit Variable Dependence: $\underline{A}x_1\underline{E}x_2[p(x_1, x_2) = 1]$

Figure 4.3

4. Scope of Variables - The head node is a logical operator and the tail is a quantified variable. This type of dependence may be necessary for certain binary logical operators such as (->, <->). For the functions v and ∠ this type of dependence is implicit in the functional dependence of the arguments. (Figure 4.4)

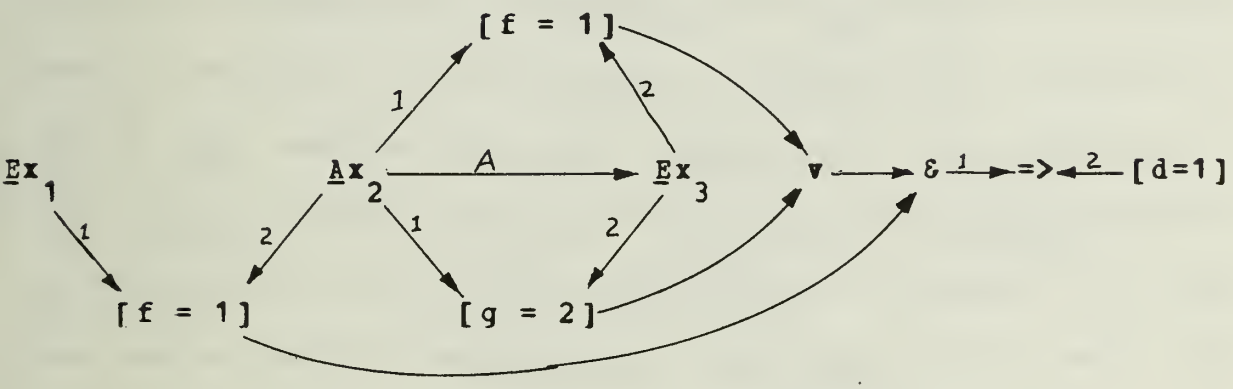


Scope of Variables: $\underline{E}x_1,x_2([p(x_1) = 1] => [g(x_2) = 1])$

Figure 4.4

The graph of a more complex decision rule is given in Figure 4.5. The value of x_3 is dependent in an unspecified way on the value of x_2 (the edge labelled A). The disjunction (v) depends on the values of x_2 and x_3, but this is clearly specified by the functional dependence of f and g on x_2 and x_3. Finally, observe that the decision operator (=>) does not

explicitly depend on the specific values of x_1, x_2, or x_3, but instead depends on the truth value of the entire premise using some set of value assignments for x_1, x_2, and x_3.



Graph Structure Example:

$$\underline{E}x_1\underline{A}x_2\underline{E}x_3((([f(x_2,x_3) = 1] \vee [g(x_2,x_3) = 2,3])$$
$$[f(x_1,x_2) = 1] \Rightarrow [d = 1]$$

Figure 4.5

5. Algorithms and Computer Implementation

A computer program INDUCE_1 has been written to find a generalization of a set of decision rules. The algorithms are described in the remaining sections of this chapter; examples of generalizations produced by the program are given in chapter 6 and a sample session with the program is given in appendix A. The program does not perform all of the transformations given in chapter 3 and does not accept the full VL_2 language given in chapter 2. The restrictions on the form of the input and a description of the output are given in the next two sections.

5.1 Input to the Program

The program accepts as input: 1) a set of decision rules, 2) a problem environment description including a set of restrictions, domain definitions, variable costs etc., and 3) a set of parameters which control certain aspects of the program operation. Decision rules, restrictions, and domain structures are entered a VL_2 type formulas in the following format:

Decision Rules

Decision rules must satisfy the following grammar:

I1. <VLRULE> ::= <CONDITION> => <DECISION>.

I2. <CONDITION> ::= <CONDITION> <SELECTOR> | <SELECTOR>

I3. <SELECTOR> ::= [<FN-SYM> (<ALIST>) = <REF>] |
[<FN-SYM> (<ALIST>)] |
[<VARIABLE> = <REF>] |
[<FN-SYM> (<ELIST>)]

I4. <ALIST> ::= <ALIST> , <VARIABLE> | <VARIABLE>

I5. <ELIST> ::= <ELIST> . <VARIABLE> | <VARIABLE>

I6. <REF> ::= <REF> , <NUMBER> |
<NUMBER> .. <NUMBER> |
<NUMBER> | *

I7. <DECISION> ::= [<FN-SYM> = <REF>]

I8. <FN-SYM> ::= string of letter

I9. <VARIABLE> ::= <FN-SYM> <NUMBER>

I10. <NUMBER> ::= string of digits.

Examples of decision rules:

$$[f(x_1,x_2) = 2,3][g(x_1) = 1..4] => [D = 1].$$

In this expression, the structure of the domain of the function g is set to interval by the program (see note 4 below). The domain of f is unchanged. All variables (e.g. x_1 and x_2) are

quantified by the operator $\underline{E}$ (distinct existential quantifier - note 3)

$$[p(x_1,x_2)\ \underline{E}\ x_2 = 2,4] \Rightarrow [D = 2].$$

In this example, the function p is assumed to have two values (i.e., it is a predicate - note 6). The selector containing p is satisfied if it has the value 1. The second selector restricts the possible values of x_2 to the set of values [2,4] (note 3).

Several observations can be made about this grammar:

1. The condition part is a single product and the decision part involves only one variable. It is assumed that one decision variable has been selected to be studied by the user. Also, the equivalence rule has been applied which splits a condition part of a formula in disjunctive normal form into a set of decision rules with condition parts which are single products (t-rules E4 and DP2).

2. Each atomic form is a function symbol with a list of single variable arguments. It is assumed that the user has converted forms such as

$$[g(f(x_i))]$$

into a form

$$[g(y_j)][p(y_j, x_i)]$$

by introducing a new predicate $p(y_j, x_i)$ for each function symbol f which is in an argument list and a variable y_j for each occurrence of the function $f(x_i)$ in an argument list. The predicate p is assumed to have the value TRUE if $y_j = f(x_i)$ and FALSE otherwise. For example:

$$[\text{shape}(\text{part}(x_1)) = 1]$$

is assumed to be transformed into e.g. an expression

$$[\text{shape}(p_1) = 1][\text{contains}(p_1, x_1)]$$

3. All variables (arguments) are assumed to be existentially quantified. Variables with the same function symbol part are assumed to have the same domain. Furthermore, variables with values from the same domain are assumed to take on distinct values from the domain. Variables may be restricted to a subrange of values by using the third option in the

selector definition. Using this method, a constant may be specified as an argument to a function.

4. If a reference of the second form is specified at least once in the input (production I6) (e.g. $[f(x_1)=2..2]$ or $[f(x_1)=2..5]$), a domain of type interval is assumed; otherwise, the domain of a variable or function is assumed to be of nominal type or tree-structured if such a structure is specified.

5. The last form of the reference uses a value (*). This specifies the entire domain of the associated function symbol. (E.g. [p=*] means that the selector is satisfied for any value of p other than the value NA.) If a selector is omitted from the decision rule entirely, the program assumes that the function has the value NA (not applicable). Such a domain value has no generalization (other than NA itself). Although NA is not specified in an input rule, it can be a valid value of of a variable.

6. If the second form of the selector is used (production I3) (e.g. $[p(x_1,x_2)]$), the program assumes that the function symbol has the value 1. This may be used to specify a TRUE value for a

predicate (predicates are treated as functions with domain [0,1]). In general, tc simplify the expressions, only positive values cf predicates are specified. The program then uses only positive instances of the predicate in the generalizations. If negative values of a predicate are desired in the generalizations, these relations should be included in the initial decision rule specification (see the arch example in chapter 6). (e.g. [ontop(p_1,p_2)] specifies that p_1 is on top of p_2; [ontop(p_1,p_2)=0] specifies that p_1 is not on top of p_2.)

7. If the fourth form of the selector is used (e.g. [p(x_1.x_2)], then the order of the arguments is assumed to be irrelevant.

Restrictions:

Restrictions must satisfy the following production:

<REST> ::= <CONDITION> - > <SELECTOR>

where the arguments of the selector part must all appear in the condition part. CONDITION and SELECTOR are the same as in the decision rule grammar. For example:

$$[\,left(x_1,x_2)\,][\,left(x_2,x_3)\,] \Rightarrow [\,left(x_1,x_3)\,].$$

Restrictions extend or modify decision rule specifications. For every occurrence of the CONDITION of the restriction in the condition part of the decision rule, the selector is added to the condition part of the decision rule if it is not already there. If the SELECTOR is already in the condition part of a decision rule, then the values of the SELECTOR replace the values in the occurrence of SELECTOR in the decision rule. Using this feature, the transitive closure of a transitive function may be calculated. A restriction which adds equivalence type predicates is included by default. Each restriction is applied to each decision rule as it is entered into the program.

Domain Generalization Structure Specification

This specification must satisfy the production:

<T-STRUC> ::= [<FN-SYM> = <REF>]=>[<FN-SYM> = <REF>]

where the two function symbols are the same and FN-SYM and REF are as in the decision rule grammar.

For example:

[s = 1,3,4] => [s = 6].

[s = 0,2] => [s = 7].

[s = 4] => [s = 8].

[s = 6,8] => [s = 9].

Entering a T-STRUC automatically sets the type of the domain for the function symbol to the structured type. Each element in the REF of the condition part of this rule must either be a leaf of the corresponding tree or have been previously defined with another T-STRUC specification.

Parameters:

Several parameters may be entered into the program. In addition to the parameters mentioned in the following sections (i.e., the MAXSTAR parameter and optimality criterion parameters), certain trace requests may be entered to follow the execution of the program.

5.2 Program Output

Several trace features are available to obtain a list of intermediate results in the program. An example of

program output is given in appendix A. Although input decision rules may be any formula which satisfies the grammar given in the previous section, the program only searches for generalizations which can be represented by a connected graph structure with functional dependence edges. The user may construct decision rules to satisfy this constraint by including new predicates which link products which have no arguments in common. (In general, it can be tested to see if a product of selectors in a decision rule has a connected graph structure by determining whether there is a partitioning of selectors into two products which have no argument in common. If there is such a partitioning, then the graph structure is not connected.)

Different generalizations may be obtained by varying the program parameters and reordering the decision rules. In general, increasing any of the MAXSTAR, ALTER, or NCONSIST parameters (described below) will cause the program to require more memory and time but the resulting generalizations may be more optimal. Rearranging the optimality criteria may also produce a different result. A higher tolerance associated with a particular cost function will reduce the selection based on that function. The default optimality criteria (or optimality cost functions) in the order of application are the following:

1. Minimize the inconsistencies of a rule with tolerance 0.30, i.e., the number of events covered by the rule which are not supposed to be covered by the rule. (this is cost function number 3 in section 5.5). This allows the program to produce consistent generalizations quickly. The high tolerance removes highly inconsistent rules while leaving selection of nearly consistent rules to the remaining cost functions.

2. Minimize the number of products in the complete generalization with tolerance 0.00 (this is cost function number 1 in section 5.5).

3. Minimize the number of selectors in each product produced by the program (function 2 in section 5.5).

4. Minimize the cost of functions in each product (function 4 in section 5.5). If costs are specified, this criterion may be moved forward. If the user wishes certain functions to appear in the resulting products, the costs for these functions may be specified (given negative cost). Similarly, functions

which are very difficult or costly to measure may be given appropriate positive costs.

5. Maximize the intersection of resulting rules (cost function 5 in section 5.5). In real situations, the separate products produced by the program may represent a large number of common input decision rules along with some peculiarities of specific situations which arise. Use of this cost function will favor the selection of a more representative result as opposed to one which describes only a particular set of situations. Appendix B contains a more detailed discussion of a similar cost function in the VL_1 system.

The program contains about 40 procedures. Five major tasks which are performed by some groups of procedures are described below.

5.3 Formation of a Complete Generalization

A generalization is found of a set of decision rules is found containing a specified value I in the decision part. Two sets of products are generated: a set P1 which contains all products in the CONDITION parts of rules with a decision value of I, and a set P0 which contains all other

products. Each product is called a c-formula (conjunctive-formula).

One c-formula E1 of F1 is selected at random and a connected-conjunctive-formula (c2-formula) is generated which is a generalization of E1, consistent with respect to the set F0, and near optimal with respect to a user defined criterion. A c-formula is connected if its graph structure representation is weakly connected by functional dependence relations. A c-formula is consistent with respect to a set of c-formulas F0 if it does not intersect with any element of the set F0 (i.e., there is no event which satisfies both c-formulas).

Once a generalization of E1 is found, it is saved in a set CQ and all elements of F1 which are covered by this generalization are removed from F1. One c-formula E1 covers another c-formula E0 if E1 is a generalization of E0. Another element of the new set F1 is selected and the procedure repeated. When there are no more elements in F1, the complete, consistent generalization of the set of c-formulas F1 is the disjunction of all c2-formulas in CQ.

5.4 Determine Cover and Intersection of 2 Formulas

Two similar procedures are decribed here. The test to determine whether a c2-formula E covers a c-formula E' is

used when E' is an element of the set F1. The test to determine whether E intersects with E' is used when E' is an element of F0 (i.e., to determine if E and E' are consistent). The procedure uses the graph structure representations of E and E' (G and G' with nodes and edges V,E,V',E' respectively). The graph G is assumed to be weakly connected. E covers E' if there is a specializing isomorphism (s-isomorphism) from G to a subgraph of G'. The reverse mapping (from a subgraph of G' to G) is called a generalizing isomorphism (g-isomorphism). E intersects with E' if there is an intersecting isomorphism (i-isomorphism) between G and a subgraph of G'. Each isomorphism from G to a subgraph of G' is a 1-to-1 correspondence between nodes and edges of G and a subset of nodes and edges of G' where the correspondence (or matching) of nodes and edges is defined as follows:

A node n of G matches a node n' of G' if:

1. They are both selector nodes or both quantified variable nodes.

and 2. If they are selector nodes, then the function symbols in both nodes are the same. If they are variable nodes, they are of the same group of variables.

and 3. With an s-isomorphism or g-isomorphism, the set of values associated with n is a generalization of the set of values associated with n'. (The sets of values may be equal.) With an i-isomorphism, the sets of values intersect. In the case of selector nodes, these values are the elements in the reference of the selector. In the case of quantified variable nodes, these values are the subranges of the variables.

An edge of G matches an edge of G' if:

1. They have the same label

and 2. The respective head nodes match and the tail nodes match.

To speed rejection, a quick scan through the nodes is made to see if there is a correspondence between nodes of G and a subset of nodes of G' (ignoring links between nodes). If there is a possible correspondence, a procedure is invoked which locates a subgraph of G' which is isomorphic to G and assigns each node of G to a corresponding node of G'. The procedure is as follows:

1. Select a starting node (n_0) of G which contains the most labelled incoming edges. (This is the selector node with the largest number of arguments.) Selecting a node of this type insures that there is a minimum of backtracking through the starting node.

2. A rooted directed a-cyclic graph G* with nodes and edges V* and E* is constructed from G by copying all nodes and edges of G to G* and assigning a direction to each edge of G* so that G* has no cycles and for each node x in [V*-n^*_0], there is a path from n^*_0 to x. (n^*_0 is the node of G* which corresponds to n_0 in G.) A traversal of the graph G* is the list of edges and nodes visited in a preorder traversal of G* with root n^*_0. A preorder traversal of a subgraph with root x visits the node x, visits each outgoing edge of x and traverses the subgraph which has as the root, the head node of the traversed edge.

3. The graph G is traversed in the order of the traversal of corresponding nodes and edges of G*. At each step of the traversal of G, a node and new edge of G' is found which match the node and edge of G. If two nodes match, they are assigned to each other

and a record of the matching nodes and edges is kept for each assignment in a backtrack list. To establish a 1-to-1 correspondence, nodes of one graph which are previously assigned can only match corresponding assigned nodes of the other graph.

4. If there is no node and edge of G' which matches a node and edge of G, the procedure backtracks to the previous nodes and edges on the backtrack list, erasing the last nodes and edges on the backtrack list and the assignments associated with the nodes. Another node and edge of G' are selected which match the last node and edge of G on the backtrack list and the traversal of G continues. If no node and edge of G' can be found at this point, then the procedure again backtracks until a new match is found or the backtrack list is exhausted.

5. If the traversal of G is complete, then G covers (intersects) G'. If the backtrack list is exhausted, then G does not cover (intersect) G'.

A feature is included which finds all subgraphs of G' which are isomorphic to G. This feature is used in section 5.7 and in adding restrictions to c-formulas.

6. If the traversal of G is complete, then the current set of assignments is the desired mapping. To find the next isomorphism, the procedure returns to step 4 assuming that the last nodes and edges on the backtrack list did not match.

5.5 Trimming a Set of c-formulas

Trimming is the process of selecting the MAXSTAR best elements of a set of c-formulas with regard to a user defined criterion. The user specifies the cost functions which are to be used, the order in which they should be applied, and the tolerance associated with each cost function. Implemented cost functions are:

1. The number of events of the current set F1 which are covered by a c2-formula. (The negative of this quantity is used to obtain a cost.) This function minimizes the number of c-formulas in CQ.

2. The number of selectors in a c2-formula. The function minimizes the number of selectors in the c-formula.

3. The number of events of F0 which intersects with a c2-formula. This function leads more rapidly to consistent c-formulas.

4. The total cost of all functions contained in a c2-formula.

5. The number of events of the original set F1 which are covered by a c2-formula. (The negative of this quantity is used to obtain a cost.) This function finds the most representative c-formulas.

A set of c-formulas is trimmed using n cost functions $(cf_1, cf_2, \ldots cf_n)$ and relative tolerance for each cost function $(tol_1, tol_2, \ldots tol_n)$. The costs are applied in the order specified by the user (cf_1 first, cf_2 second, etc). For each cost function cf_i, the MAXSTAR best c2-formulas along with all c2-formulas equivalent in cost to the MAXSTAR best c-formulas are passed to the evaluation using the next cost function cf_{i+1}. Other c2-formulas in the set of c-formulas are discarded. With the last specified cost function (cf_n), only the MAXSTAR best c-formulas are retained.

For each cost function cf_i, i=1,2,...,n, equivalence of two c-formuas in cost is defined using an absolute

tolerance (AT_i). Suppose the set of c-formulas P is composed of a list $p_1, p_2, \ldots p_m$. After values for cost function cf_i have been evaluated $cf_i(p_j)$ for each c-formula p_j, the maximum and minimum cost function values are determined $cf_i(p_{max})$ and $cf_i(p_{min})$. An absolute tolerance (AT_i) is calculated using the user specified tolerance tol_i as follows:

$$AT_i = tol_i * (cf_i(p_{max}) - cf_i(p_{min}))$$

The MAXSTAR c-formulas of least cost are determined and the list reordered $(p_1, p_2, \ldots, p_{MAXSTAR}, \ldots, p_m)$. If i<n then c2-formulas which are not equivalent in cost to $p_{MAXSTAR}$ are discarded.

$$P = P - [p_j : cf_i(p_j) - cf_i(p_{MAXSTAR}) > AT_i]$$

If i=n, (the last cost function) then only the MAXSTAR best c2-formulas are retained.

$$P = P - [p_j : j > MAXSTAR].$$

The set of c2-formulas which remains is the desired trimmed set of c2-formulas.

5.6 Formation of a Set of Consistent Generalizations

A star (denoted by MQ) is formed which covers E1. (A star which covers E1 is a set of consistent c2-formulas which cover E1.) The procedure begins by forming a partial star (P) which contains a set of c-formulas each consisting of one selector of E1. (The partial star may contain c2-formulas which are not consistent with respect to F0.) This partial star is trimmed according to the user supplied optimality criterion. The conjunction in each c-formula which remains after trimming is multiplied by each selector of E1 which is directly connected to it to form a new partial star. Consistent c2-formulas are placed in MQ. The partial star is again trimmed and new selectors added to each product until the desired set MQ of c2-formulas is obtained. Several paramenters control the sizes of sets in this procedure:

MAXSTAR - the number of c-formulas in a partial star after trimming

NCONSIST - the minimum number of consistent c2-formulas which must be in MQ

ALTER - the maximum number of new alternatives which may be formed by adding selectors to an element of a partial star.

In the following discussion, equivalence type selectors (i.e., selectors of the form [$f(x_1,x_2)$=same]) are treated differently from selectors involving a function symbol and a set of values in the reference.

1. A partial star P is formed which contains all selectors of E1 with unary functions.

2. P is trimmed to contain only the best MAXSTAR c2-formulas. Consistent c-formulas are placed into MQ. If fewer than NCONSIST elements are in MQ, then step 3 is executed. Otherwise, the AQ procedure is applied to the elements of MQ (as described in section 5.7).

3. A new partial star P' is formed from the old one (P). For each element p_i in P, a list of all variables (i.e., arguments of selectors of p_i) is formed. All arguments of equivalence type selectors which occur in the corresponding selector of E1 are also included in the list.

4. A list of all selectors of E1 which are not already in p_i and which have at least one argument in the

variable list (found in step 3) is created. If there are more than ALTER elements in this list, the best ALTER selectors are retained (using as a criterion the cost of functions in the selectors).

5. For each of these selectors, a new c2-formula is formed which contains the original selectors in p_i and the new selector. If the new c-formula contains an equivalence selector with only one argument, then the new c-formula is discarded; otherwise, it is placed in P'. Steps 2 through 5 are repeated, setting P = P' in step 2 until NCONSIST elements are in MQ or until no new elements are in the new partial star P'.

5.7 Extending the References of a Consistent c2-formula

Each consistent c2-formula of MQ (obtained in section 5.6 step 2) contains an alternative, near optimal conjunction of selectors of E1 which distinguishes E1 from any c-formula of F0. Using some methods developed for the program AQ7, the reference of each of these selectors may be generalized to obtain a consistent c2-formula which will possibly cover possibly more c-formulas of F1.

Given a graph G of a conistent c2-formula mq in MQ, a <u>c-structure</u> is created (G*) by replacing all references of

nodes of G with * (the complete set of values for the function in the selector). The nodes of G* are enumerated n^*_i (i=1,2,...,m) and a VL_1 system is created with each VL_1 variable x_i related to a node n^*_i of G*. The domain (denoted D_i) of variable x_i is the same as the domain of the function or variable in the node n^*_i.

The VL_1 events space may be defined:

$$E = D_1 \times D_2 \times \ldots \times D_n.$$

Two sets of VL_1 complexes L and L' are formed from the events of the current set F1 and the set F0 respectively. Individual complexes in these sets are denoted l_i and l'_i. Each compelex covers a set of points in the space E. For each element of F1 and F0 all isomorphisms from the c-structure G* to the graph representation G of a c-formula in F1 or F0 are determined. For each isomorphism obtained from F1 and F0, a VL_1 complex is created (l_i or l'_i). Denoting the value sets of the nodes in a subgraph of G which is isomorphic to G* as $R_1, R_2, \ldots, R_m$, the corresponding VL_1 complex may be written:

$$[x_1 = R_1][x_2 = R_2] \ldots [x_m = R_m].$$

This complex covers the VL_1 events:

$$R_1 \times R_2 \times \ldots \times R_m$$

in the event space E.

First, the complex l_1 which results from extracting the values from the nodes of the graph of mq is generated. Then all other isomorphisms from G* to c-formulas of F1 are determined and a complex added to L for each isomorphism which results in a new complex that is not already in L. The set L' is created in a similar manner by generating all distinct complexes resulting from isomorphisms from G* to c-formulas of F0.

Since the c-formula mq is consistent with regard to F0, the complex l_1 is disjoint from all complexes in L'. (That is, there is no point in the VL_1 event space E which is in both l_1 and a complex of L'.) A near optimal extension of l_1 against L' in E may be calculated using a version of the AQ7 program. The best complex in this extension (l_q) is calculated according to a user defined criterion. The l_q complex is converted to a c-formula by replacing the value set of each node n^*_i of G* by the reference of the selector with variable x_i in l_q. This c-formula is consistent with respect to the set F0. (This is evident since, if there were a VL_1 event which satisfied both the c-formula from l_q and a c-formula of F0, then one could find a VL_1 complex -- using

an isomorphism between G* and the c-formula of F0 -- which intersected with l_1 and L'. But since l_1 and L' are disjoint, this can not happen.)

The cost function for the AQ7 procedure computes the cost of a complex. Cost functions may be selected from the following:

1. The number of elements in L which are covered by a complex but not covered by any previous l_q. This is the AQ7 counterpart to the cost function 1 in section 5.5) (Use the negative of this value to get a cost.)

2. The number of selectors in a complex (the AQ7 counterpart to function 2 in section 5.5).

3. The number of elements of L covered by a complex which are associated with different events of F1.

4. The total cost of variables which appear in a complex (i.e., the cost of functions or variables in associated nodes of G*). This is the counterpart of the function 4 in section 5.5.

5. The total number of events in L covered by a complex.

6. The number of events in L' covered by a complex (the AQ7 counterpart to function 3 in 5.5).

Trimming is done in the manner described above for c-formulas (sec 5.5). A MAXSTAR parameter is specified for this procedure and the l_q is selected from the extension using a MAXSTAR value of 1. Appendix B gives a more complete description of the AQ7 procedure.

5.8 Adding New Functions and Predicates to c-formulas

The program currently allows three types of new functions to be added to existing c-formulas: 1) global descriptors (meta functions) which count the frequency of occurrence of selectors with unary functions; 2) equivalence type predicates (of the form [$f(x_1,x_2)$=same] i.e., the value of f is the same for x_1 and x_2); 3) extremity type predicates ([lst-$f(x_1)$] or [mst-$f(x_1)$]) which indicate that the argument x_1 is at one end of a sequence of binary predicates. Functions are added at the user's discretion.

Meta Selectors - There are two types of meta functions currently calculated and added to a c-formula as a meta selector: One type (#PT(f=a) where f is an atomic function and a is a value in D(f)), counts the

number of times a particular selector ([f(..)=a]) appears in a c-formula. The second type (FORALL(f=a)) is a predicate which is true if a function assumes only one value in a c-formula and false otherwise.

For each atomic function-reference pair which appears in any c-formula, a meta selector is added to the c-formula which has a meta function in the referee. For example, a c-formula

$$[tx(x_1)=1][tx(x_2)=1][sh(x_1)=1][sh(x_2)=0]$$

generates the four meta selectors:

[#PT(tx=1)=2] - the number of parts with tx=1 is 2

[#PT(sh=0)=1] - the number of parts with sh=0 is 1

[#PT(sh=1)=1] - the number of parts with sh=1 is 1

[FORALL(tx=1)] - all parts have tx=1.

Since the number of such selectors may be quite large, the list of meta functions is trimmed to a small set. The size of the set is determined by a parameter METATRIM supplied by the user using as criteria the degree to which a value of the new function will separate the sets F1 and F0. For each

meta function-value combination (meta selector) which is generated, the number of c-formulas of F1 and F0 which satisfy the selector is calculated. Associated with the meta function (mf) are two numbers: F1COV and F0COV. F1COV is the maximum number of c-formulas of F1 covered by a meta selector arising from the meta function mf. F0COV is the number of c-formulas of F0 which are covered by the meta selector which gave the highest F1COV value.

The list of possible meta functions is trimmed to METATRIM remaining meta functions by sorting in descending order, the list of meta functions according to the primary field F1COV and the secondary field F0COV and selecting the first METATRIM functions from this list. The meta selectors which result from applying each of the selected meta functions to each c-formula are automatically appended to the c-formulas and carried with each c-formula during the generalization process. Setting METATRIM to 0 bypasses the entire meta selector generation process.

Equivalence Predicates - These predicates may have arbitrarily many arguments whose order is

irrelevant. They are calculated by scanning all selectors in each c-formula for sets of 2 or more selectors with unary functions which have the same atomic function and reference. Such a set of selectors is said to be equivalent and a new predicate is created which contains an argument from each of the atomic forms of all equivalent selectors. For example, a c-formula of the form:

$$[s(x_1)=1][s(x_2)=1][s(x_3)=1][s(x_4)=2][s(x_5)=2]$$

leads to the creation of predicates:

$$[s(x_1 \cdot x_2 \cdot x_3)=\text{same}][s(x_4 \cdot x_5)=\text{same}]$$

Extremeties Predicates - These are unary predicates which represent the ends of a sequence of selectors with the same binary predicate:

$$[p(x_1,x_2)][p(x_2,x_3)] \cdots [p(x_{i-1},x_i)]$$

where, in the sequence, if $i>2$ then the first argument of the j-th selector ($1<=j<i$) is the same as the second argument of the j+1-st selector. Also, the variables x_1 (and x_i) do not appear as the second (or first) argument of another selector with atomic

function p. The new predicates formed for the arguments x_1 and x_i of such a sequence are written:

$$[\text{MST-p}(x_1)] \text{ and } [\text{LST-p}(x_i)].$$

For example, the c-formula:

$$[\text{ontop}(x_1, x_2)][\text{ontop}(x_1, x_3][\text{ontop}(x_2, x_4)]$$

would give rise to the new predicates:

$$[\text{MST-ontop}(x_1)][\text{LST-ontop}(x_3)][\text{LST-ontop}(x_4)].$$

6. Examples of Decision Rule Generation using INDUCE_1

The program INDUCE_1 is written in PASCAL for either a CDC-CYBER 175 or a DEC-10 computer. (The two versions differ in syntax and certain input output specifications.) They contain approximately 3300 statements and occupy 34K octal 60-bit words of CDC memory or 18K 36-bit words of DEC-10 memory. The examples in this chapter were run on the CYBER 175 using a field length of 130K octal words to include both the program and data. Execution time estimates which are given below are for the CYBER and should be accurate to within 30%. One example was run on both the CYBER 175 and a DEC-KI10 to obtain a time comparison between machines. Execution time on the DEC-10 was 8 times larger than on the CYBER.

For each example, a sample of the input decision rules is listed along with tables describing function domains and program parameters. The resulting generalizations are given as decision rules in disjunctive normal form. The set of objects associated with a description below is given as the value of the decision part of the rule (e.g. for a set O_1, the associated decision part is [d=1]). With some examples, several alternatives were produced by the program. Each alternative which was produced is given as a separate

decision rule. The various symbols contained in these tables are defined below:

Program Parameters:

Parameter	Description
AQMAXSTAR	the MAXSTAR parameter for the AQ7 procedure. This parameter specifies the number of alternative complexes retained in each partial star in the AQ7 procedure (see appendix B).
LQST	the LQST parameter in the AQ7 procedure. If this parameter is set, then each l_q which is generated in the AQ7 procedure is stripped so that the reference in the selector for each variable in the l_q complex contains the minimum number of elements under the constraints of the domain structure while covering the same number of events of L (see appendix B).
AQCRIT	the list of cost functions used in the AQ7 procedure in the order of application (the minus sign indicates that the quantity

which is calculated by the function is negated to obtain the cost of the complex) (see section 5.7).

AQTOLERANCE the tolerance associated with each cost function. If this figure is less than 1, then it is a relative cost, otherwise it is an absolute tolerance (see section 5.5).

AQNF the number of consistent generalizations which are created in the procedure which forms consistent generalizations (i.e., the minimum number of elements in the star MQ) (see section 5.6).

ALTER the number of alternatives which are retained in a partial star (see section 5.6).

VLCRIT the list of cost functions used to trim a partial star of c-formulas in the order of application (a minus sign indicates that the quantity which is caluculated by the function is negated to obtain the cost of the c-formula) (see section 5.5).

VLTOLERANCE the tolerance associated with each cost function. A value which is less than 1 is assumed to be a relative tolerance; otherwise, it is an absolute tolerance (see section 5.5).

METATRIM the number of meta selectors of the type FORALL and #PT(number of parts) which are added to each formula.

EXTMTY the predicates of the type [MST-f(x_i)] and [LST-f(x_i)] were added to input decision rules.

EQUIV the predicates of the type [f(x_1.x_2)=same] were added to the input decision rules. (These correspond to the predicates [EQ_f(x_1.x_2)] used above)

Domain descriptions:

Column	Description
NAME	the name of the function
NARG	the number of arguments of the function

TYPE	the domain structure: 1-nominal, 2-interval, 3-tree structured.
COST	the cost of the function
MIN	the minimum value in the domain of the function
MAX	the maximum value in the domain of the function
STRUCTURE	the allowable generalizations in tree structure domains

Meta Selectors:

Column	Description
MS	the meta selector number associated with this function
TYPE	either #PT (number of parts) or FORALL
FUNCTION	the associated function and value (i.e., the associated selector ignoring arguments e.g. FORALL TX = 0 is a predicate which

has the value 1 if all parts have a texture of 0 and 0 otherwise)

F1COV — the number of c-formulas in F1 covered by one value of the meta function.

F0COV — the number of c-formulas in F0 which are covered by the meta selector associated with the value F1COV above (see section 5.8).

6.1 Figures Example (EX 1)

The objects in figure 6.1 (from [Michalski 1977]) are examples of three classes of situations: O_1, O_2, and O_3, each class containing 3 examples. The figures are expressed as VL_2 formulas using 6 descriptors below. The values of descriptors are translated to integers as follows:

Descriptor	Structure	Domain
ontop	nominal	0-false, 1-true
inside	nominal	0-false, 1-true
next	nominal	0-false, 1-true
size	interval	0-small, 1-medium, 2-large

tx(texture)	nominal	0-clear, 1-shaded
shape	tree-struc	0-triangle, 1-circle,
	1,5 => 8	2-rectangle, 3-diamond
	0,2,3,7=>9	4-open-top, 5-ellipse
		6-open bottom, 7-square

The input decision rule for the first object in O_1 has the following form:

[ontop(p_1,p_2)] [ontop(p_2,p_3)] [size(p_1)=1] [size(p_2)=1]
[size(p_3)=2] [tx(p_1)=0] [tx(p_2)=1] [tx(p_3)=0] [shape(p_1)=3]
[shape(p_2)=1] [shape(p_3)=4] [inside(p_2,p_3)]=>[d=1].

with domain structure:

[shape=1,5] => [shape=8].

[shape=0,2,3,7] => [shape=9].

The resulting domain table is:

NAME	NARG	TYPE	COST	MIN	MAX	STRUCTURE
FORALL	0	1	0	0	1	
#PT	0	2	0	0	0	

shape	1	3	0	0	9 1 5=> 8; 0 2 3 7=> 9;
ontop	2	1	0	1	1
p	0	1	0	1	4
p_1	0	1	0	1	4
p_2	0	1	0	1	4
p_3	0	1	0	1	4
p_4	0	1	0	1	4
size	1	2	0	0	2
tx	1	1	0	0	1
inside	2	1	0	1	1
d	0	1	0	1	3
next	2	1	0	1	1

Generalized decision rules using no new functions (8 seconds)

The program was run with the above decision rules several times with different parameters for each run. The first run through the program used modest values for the parameters AQMAXSTAR, ALTER, VLMAXSTAR and NCONSIST and default values and ordering of criteria with no new functions added. The parameter values and resulting generalizations are given below:

Parameters:

AQPARMS VLPARMS

AQMAXSTAR = 4 LQST NCONSIST = 4 ALTER = 4 VLMAXSTAR = 4

AQCRIT	AQTOLERANCE	VLCRIT	VLTOLERANCE
-1	0	3	0.30
2	0	-1	0
4	0	2	0

NBR OF CRIT: AQNF = 2 VLNF = 3

NEW FNCTNS: METATRIM = 0

For the set O_1, the program discovered one product which covers all elements of O_1 but no element of O_2 or O_3. The product represented as a decision rule is:

$[ontop(p_1,p_2)][size(p_1)=1][size(p_2)=1][tx(p_1)=0] => [d=1]$.

(If there is a clear, medium size part on top of a medium size part, then make decision d=1.)

Using a different set of parameters (NCONSIST=2), the description (which is less optimal according to the criteria specified) is obtained:

$[ontop(p_1,p_2)][shape(p_1)=9][shape(p_2)=9] \vee [shape(p_1)=3] => [d=1]$

The description of the set O_2 produced several alternative generalizations which describe one or two parts

of O_2 but only one generalization which covered all 3 parts in O_2:

$[ontop(p_1,p_2)][size(p_2)=1][shape(p_1)=8]=>[d=2]$

(there is an ellipsoid on top of a part with medium size)

there was no single product which covered all objects in O_3. The program found one generalization which covered two examples of O_3, namely:

$[shape(p_1)=5][tx(p_1)=1]=>[d=3]$

(there is a shaded ellipse)

In covering the second object in O_3, the program found a number of combinations of shape, texture and size of parts of the example of O_3. The rule which was selected is:

$[ontop(p_1,p_2][shape(p_1)=2][tx(p_2)=1]=>[d=3]$

(there is a rectangle on top of a shaded part)

giving a final description of the set O_3:

$[shape(p_1)=5][tx(p_1)=1] \vee [ontop(p_1,p_2)][shape(p_1)=2]$
$[tx(p_2)=1]=>[d=3]$

Generalization with EXTNTY type predicates (8 seconds)

In the second application of the program to the example in figure 6.1, the EXTNTY type predicates were added to each input decision rule. The only difference in parameters is that these new predicates are added to the input decision rules. The results are the following:

For the set O_1, there were two alternative generalizations each of which cover all three parts in O_1:

$[shape(p_1)=9][tx(p_1)=0][MST\text{-}ontop(p_1)] \Rightarrow [d=1]$

(the top part is a clear polygon)

$[size(p_1)=1][tx(p_1)=0][MST\text{-}ontop(p_1)] \Rightarrow [d=1]$

(the top part has medium size)

Only one product was found to cover the set O_2:

$[shape(p_1)=8][MST\text{-}ontop(p_1)] \Rightarrow [d=2]$

(the top part is an ellipsoid)

There is still no single product which covers all objects in O_3 however, the description of the second object in O_3 was somewhat simplified:

$[shape(p_1)=5][tx(p_1)=1] \vee [shape(p_1)=2][MST\text{-}ontop(p_1)] \Rightarrow [d=3]$

Generalizations with meta selectors (9 seconds)

The next pass of the program included the three meta selectors which were determined by the program to be the best for describing each set of objects (using as a criteria the values of F1COV and F0COV). The results along with the interpretation of the selected meta selectors are:

[ms1=2]=>[d=1]
(there are 2 clear parts)

The selected meta-selectors are:

MS	TYPE	FUNCTION			F1COV	F0COV
1	#PT	tx	=	0	3,	0
2	#PT	size	=	1	3,	3
3	#PT	shape	=	6	3,	5

The three meta selectors count the number of parts with each texture and the number of square parts.

The set O_2 was covered by the disjunction of two selectors:

[ms3=1] v [ms3=3]=>[d=2]
(there are either one or 3 clear parts)

The selected meta-selectors are:

MS	TYPE	FUNCTION			P1COV	P0COV
1	#PT	shape	=	7	3,	4
2	#PT	shape	=	6	3,	5
3	#PT	tx	=	0	2,	0

Recall that #PT has an interval domain structure. Giving #PT a nominal domain structure, the rule is:

[ms3=1,3]=>[d=3]

The most interesting simplification is with the set O_3. Two alternatives were discovered:

[ms1=0]=>[d=3]

[ms2=1]=>[d=3]

(all parts have shaded texture)

The selected meta-selectors are:

MS	TYPE	FUNCTION			P1COV	P0COV
1	#PT	tx	=	0	3,	0
2	FORALL	tx	=	1	3,	0
3	#PT	shape	=	5	2,	2

In summary, alternative descriptions for each set of objects are:

[#PT(tx=0)=2]=>[d=1] or

[shape(p_1) =9][size(p_1) =1][MST-ontop(p_1)]=>[d=1] or

[shape(p_1) =9][tx(p_1) =0][MST-ontop(p_1)]=>[d=1]

(there are 2 clear parts or the top part is a clear polygon or a polygon of medium size)

[shape(p_1) =8][MST-ontop(p_1)]=>[d=2] or

[#PT(tx=0]=1,3]=>[d=2]

(the top part is an ellipsoid or there are one or three clear parts)

[FORALL(tx=1)]=>[d=3] or

[#PT(tx=0) =0]=>[d=3]

(all parts are shaded)

Descriptive generalizations (10 seconds each set)

If the program is given the description of only one set of objects at a time, then a descriptive generalization of sorts may be found by adjusting some parameters. The significant modification of parameters is that the cost function 2 (number of selectors) is replaced by -2 (i.e., the more selectors, the lower the cost). In addition, a restriction was added to the predicate 'ontop' to implement the transitive closure for this predicate:

[ontop(p_1,p_2)][ontop(p_2,p_3)] => [ontop(p_1,p_3)].

The resulting descriptions contain a product which covers all objects in a set and contains as many selectors as possible. Of course, since the descriptions of the other two sets of objects is not included, the descriptions are not discriminant but they are complete.

AQPARMS			VLPARMS	
AQMAXSTAR = 4	LQST	NCONSIST = 70	ALTER = 3	VLMAXSTAR = 3

AQCRIT	AQTOLERANCE	VLCRIT	VLTOLERANCE
-1	0	3	0.30
-2	0	-1	0
4	0	-2	0

NBR OF CRIT: AQNF = 2 VLNF = 3

NEW FNCTNS: METATRIM = 3 EXTMTY

For each set, the products with the maximum number of selectors which the program found to be common to all objects of the set are in the following decision rules:

[ontop(p_1,p_2)][size(p_1)=1][tx(p_1)=1][shape(p_1)=9][tx(p_2)=0]
[MST-ontop(p_1)][size(p_2)=1,2][shape(p_2)=2,4,5][tx(p_2)=0]
[LST-ontop(p_2)][ms1=0][ms2=2][ms3=2]=>[d=1]

The selected meta-selectors are:

MS	TYPE	FUNCTION			F1COV	F0COV
1	#PT	shape	=	6	3,	0
2	#PT	size	=	1	3,	0
3	#PT	tx	=	0	3,	0

(The bottom part is clear and has size medium or large. The top part is a clear polygon, has medium size and there are no parts with an open bottom, 2 parts of medium size, and 2 clear parts.)

[ontop(p_1,p_2)][ontop(p_1,p_3)][size(p_1)=1,2][size(p_2)=1,2]
[shape(p_1)=8][shape(p_2)=0,1,2,4][LST-ontop(p_2)]
[ms1=0][ms2=0][ms3=0,1]=>[d=2]

The selected meta-selectors are:

MS	TYPE	FUNCTION			F1COV	F0COV
1	#PT	shape	=	6	3,	0
2	#PT	shape	=	7	3,	0
3	#PT	shape	=	5	2,	0

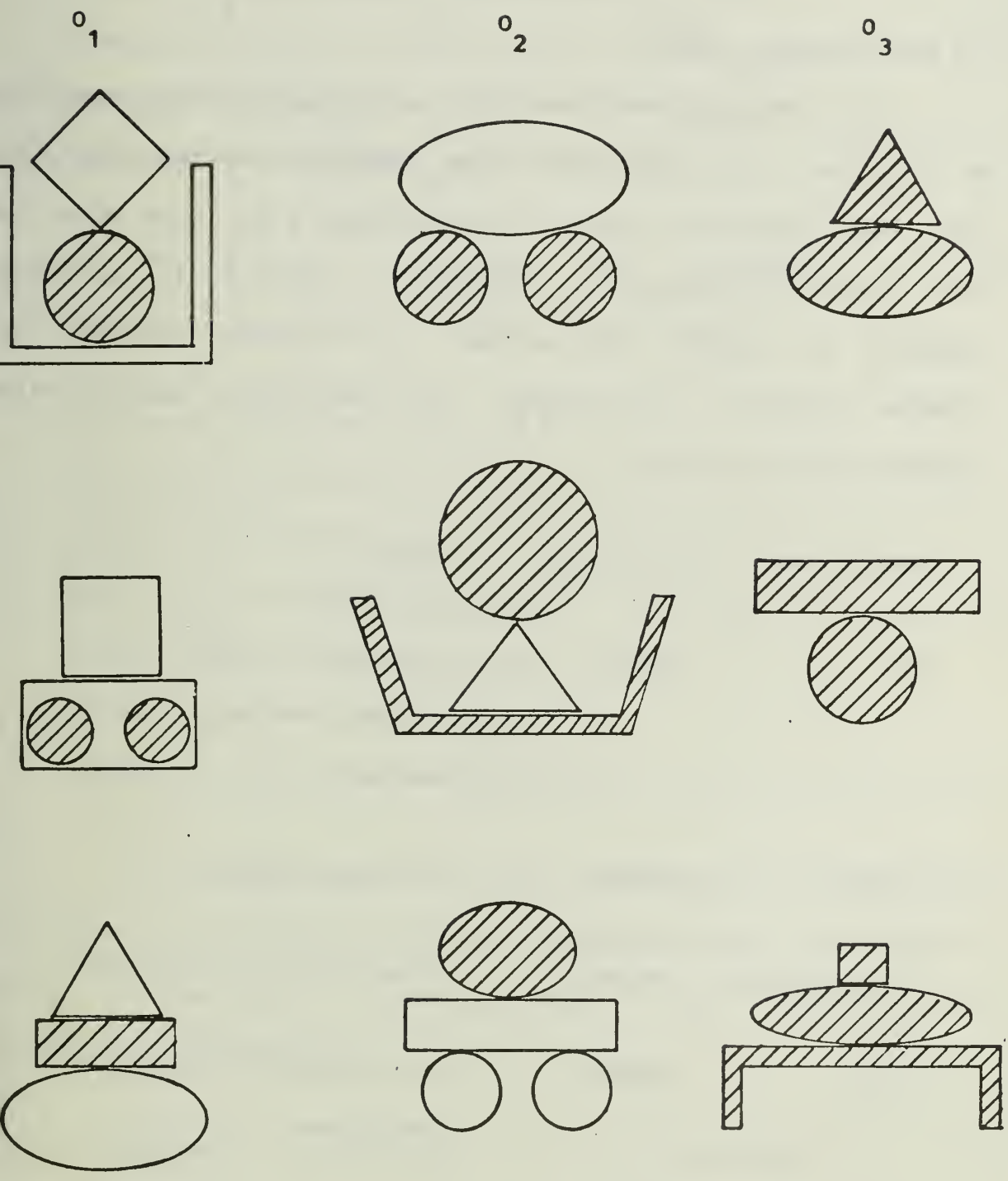
(The top part is an ellipsoid which is not small and is on top of two parts. The bottom part is of medium or large size. Also, there are no square or open bottom parts and no more than 1 ellipse.)

[ontop(p_1,p_2)][size(p_1)=0,1][size(p_2)=1,2][tx(p_1)=1]
[tx(p_2)=1][MST-ONTOP(P_1)][SHAPE(P_1)=9]
[LST-ontop(p_2)][ms1=0][ms2=1][ms3=0,1]=>[d=3]

The selected meta-selectors are:

MS	TYPE	FUNCTION			F1COV	F0COV
1	#PT	tx	=	0	3,	0
2	FORALL	tx	=	1	3,	0
3	#PT	shape	=	6	2,	0

(There are at least two parts both shaded with the top part being a polygon of small or medium size and the bottom part being of medium or large size. All parts are shaded and there is not more than 1 open bottom part.)



Figures for Example EX 1

Figure 6.1

6.2 Arch Example (EX 2)

The objects in figure 6.2 are examples which describe an arch much like the arch which Winston describes [Winston 70]. Some additional shapes and relations have been added to give a more interesting example. The set A_1 includes examples of arches and the set A_2 contains examples of objects which are not arches. The description used in this example are given below:

Descriptor	Structure	Domain
rel	nominal	0-on top, 1-under 2-left, 3-right 4-other
touch	nominal	0-false, 1-true
or(ientation)	nominal	0-horizontal, 1-vertical
shape	nominal	0-rectangle, 1-triangle 2-ellipse

The decision rule for the first arch in A_1 as given to the program is:

$[rel(p_1,p_2)=0]\ [rel(p_1,p_3)=0]\ [rel(p_2,p_3)=2]$

$[touch(p_1,p_2)=1]\ [touch(p_1,p_3)=1]\ [touch(p_2,p_3)=0]$

$[or(p_1)=0]\ [or(p_2)=1]\ [or(p_3)=1]\ [shape(p_1)=0]$

$[shape(p_2)=0]\ [shape(p_3)=0] \Rightarrow [d=1].$

Restrictions describing the character of the functions rel and touch are added to each rule:

$[rel(p_1,p_2)=4] \Rightarrow [rel(p_2,p_1)=4].$

$[rel(p_1,p_2)=0] \Rightarrow [rel(p_2,p_1)=1].$

$[rel(p_1,p_2)=1] \Rightarrow [rel(p_2,p_1)=0].$

$[rel(p_1,p_2)=2] \Rightarrow [rel(p_2,p_1)=3].$

$[rel(p_1,p_2)=3] \Rightarrow [rel(p_2,p_1)=2].$

$[touch(p_1,p_2)=1] \Rightarrow [touch(p_2,p_1)=1].$

$[touch(p_1,p_2)=0] \Rightarrow [touch(p_2,p_1)=0].$

The program was run twice with these examples: once to find an optimal discriminant generalization, and again to find a descriptive generalization. The parameters and results are as follows:

AQPARMS VLPARMS

AQMAXSTAR = 2 LQST NCONSIST = 12 ALTER = 1 VLMAXSTAR = 1

AQCRIT	AQTOLERANCE	VLCRIT	VLTOLERANCE
-1	0	3	0.30

-2	0	-1	0
4	0	-2	0

NBR OF CRIT: AQNF = 2 VLNF = 3

NEW FNCTNS: METATRIM = 0 EQUIV

Discriminant description of set A_T: (12 seconds)

$[rel(p_1,p_2)=0]\ [touch(p_1,p_2)=1]\ [touch(p_1,p_3)=1]$
$[touch(p_2,p_3)=0]\ [or(p_3)=1]\ [or(p_2,p_3)=same]$
$[shape(p_2,p_3)=same]=>[d=1]$

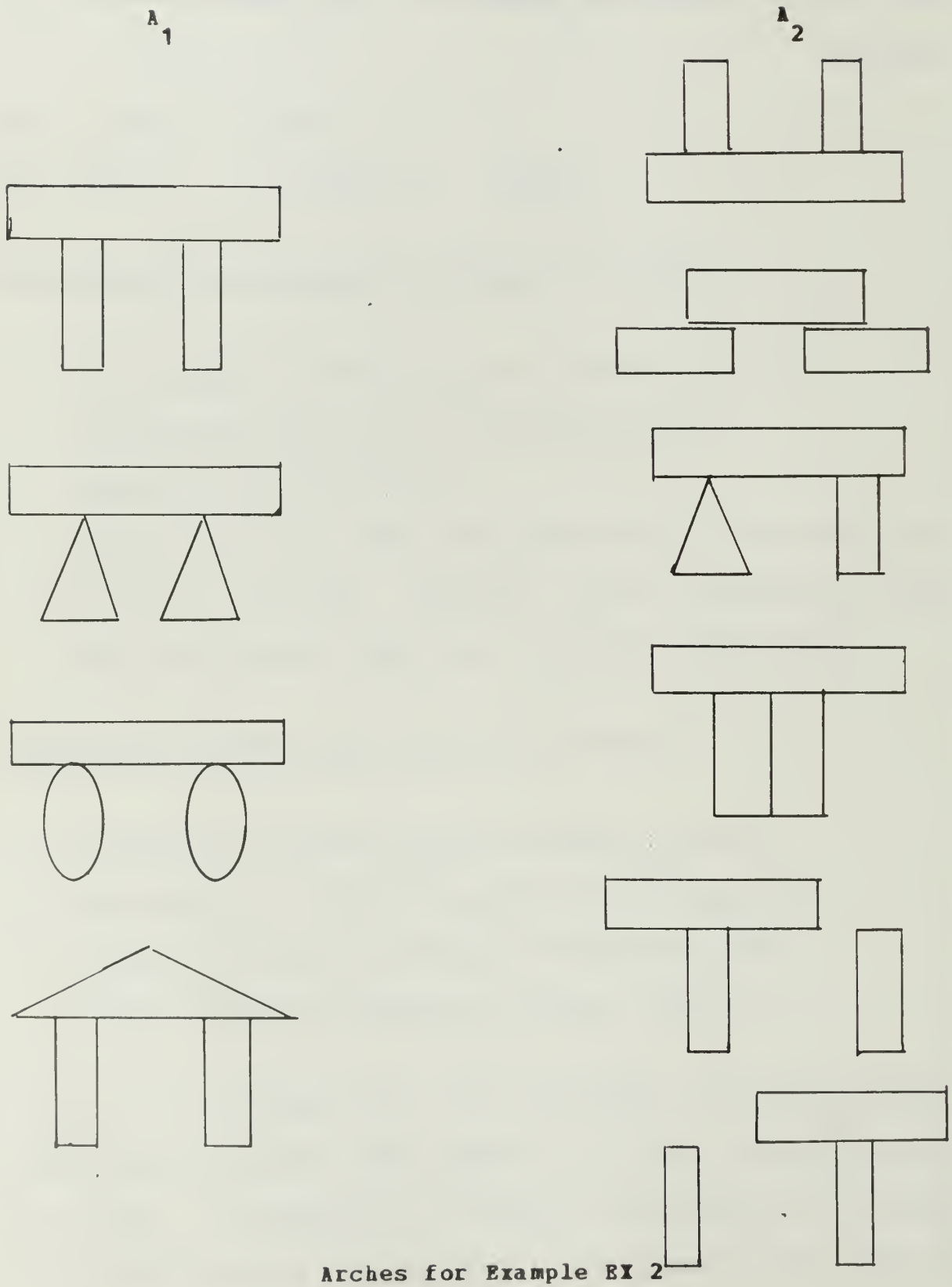
(There is a part which is touching two others and on top of one. The other two parts are not touching but they both have the same shape and a vertical orientation).

Descriptive decision rule for A_T: (2 seconds)

$[rel(p_1,p_2)=0]\ [rel(p_1,p_3)=0]\ [rel(p_2,p_3)=2,3]$
$[touch(p_1,p_2)=1]\ [touch(p_1,p_3)=1]\ [touch(p_2,p_3)=0]$
$[or(p_1)=0]\ [or(p_2)=1]\ [or(p_3)=1]\ [shape(p_1)=0,1]$
$[or(p_2,p_3)=same]\ [shape(p_2,p_3)=same]\ =>[d=1]$

It so happens that the descriptive decision rules are also discriminant in this instance but this is normally not the case. In comparison to Winston's description of an arch, the results relate very closely to his result, namely that an

arch has two supporting members of a top piece which are not touching.



Arches for Example EX 2

Figure 6.2

6.3 Trains (EX 3)

The trains in figure 6.3 (from [Larson Michalski 1977]) represent two sets of trains, one set going west and the other set going east. There are 12 descriptors which seem relevant to the situation in figure 6.3:

Descriptor	Structure	Domain
ncars	interval	[3:5]
nwhl (#whl/car)		interval [2,3]
ln (length)	interval	0-short, 1-long
cshape	tree-struc	0-open rctngl, 1-open trap.
	0,1,2,5=>10	2-U-shaped, 3-hexagon
	3,4,6,7,8,9=>11	4-ellipse, 5-dbl open rctngl
		6-closed rctngl, 7-jagged top
		8-sloping top, 9-locomotive
		10-open top, 11-closed top
npl (#pts/load)		interval [0:3]
lshape	nominal	0-circle, 1-hexagon
		2-triangle, 3-rectangle
if (in front)	nominal	0-false, 1-true
ccont	nominal	0-false, 1-true
lcont	nominal	0-false, 1-true
t (train)	nominal	[1]
car	nominal	[1:5]

lod nominal [1:5]

The variable (t) and the predicate ccont (train contains car) are introduced to form weakly connected graph structure representations of input decision rules. The description of each train contains about 60 selectors so only the description of the first train is given below:

[ncar(t_1)=5] [ccont(t_1,car_1)=1] [ccont(t_1,car_2)=1]
[ccont(t_1,car_3)=1] [ccont(t_1,car_4)=1] [ccont(t_1,car_5)=1]
[if(car_1,car_2)=1] [if(car_2,car_3)=1] [if(car_3,car_4)=1]
[if(car_4,car_5)=1] [loc(car_1)=1] [loc(car_2)=2]
[loc(car_3)=3] [loc(car_4)=4] [loc(car_5)=5] [nwhl(car_1)=2]
[nwhl(car_2)=2] [nwhl(car_3)=2] [nwhl(car_4)=3] [nwhl(car_5)=2]
[ln(car_1)=1] [ln(car_2)=1] [ln(car_3)=0] [ln(car_4)=1]
[ln(car_5)=0] [cshape(car_1)=9] [cshape(car_2)=0]
[cshape(car_3)=8] [cshape(car_4)=0] [cshape(car_5)=0]
[npl(car_1)=0] [npl(car_2)=3] [npl(car_3)=1] [npl(car_4)=1]
[npl(car_5)=1] [lcont(car_2,lod_1)=1] [lcont(car_2,lod_2)=1]
[lcont(car_2,lod_3)=1] [lcont(car_3,lod_4)=1]
[lcont(car_4,lod_5)=1] [lcont(car_5,lod_6)=1] [lshape(lod_1)=3]
[lshape(lod_2)=3] [lshape(lod_3)=3] [lshape(lod_4)=2]
[lshape(lod_5)=1] [lshape(lod_6)=0] ->[d=1].

The parameters and discriminant generalizations from this example are now presented.

AQPARMS VLPARMS

AQMAXSTAR = 4 LQST NCONSIST = 8 ALTER = 8 VLMAXSTAR = 1

AQCRIT	AQTOLERANCE	VLCRIT	VLTOLERANCE
-1	0	3	0.30
2	0	-1	0
4	0	2	0

NBR OF CRIT: AQNF = 2 VLNF = 3

NEW FNCTNS: METATRIM = 0

The discriminant generalizations obtained were the following:
(10 seconds)

$[cshape(car_1)=11][ln(car_1)=0]=>[d=1]$

(there is a short, closed top car)

$[ncar(t_1)=3]$ v $[cshape(car_1)=7]=>[d=2]$

(there are three cars or a car with a jagged top)

Some other interesting generalizations which were obtained by using different parameters are:

$[ncars(t_1)=4,5][lshape(lod_1)=2]=>[d=1]$

(If there are at least 4 cars and one car has a load which is a triangle then the train is going east.)

[loc(car_1) =4][cshape(car_1) =6] v

[loc(car_1) =5][cshape(car(1) =0]=>[d=1]

(If the fourth car is a closed rectangle or the fifth is an open rectangle then the train is going east.)

[#PT(ln=1) =2][FORALL(nwhl=2)] v

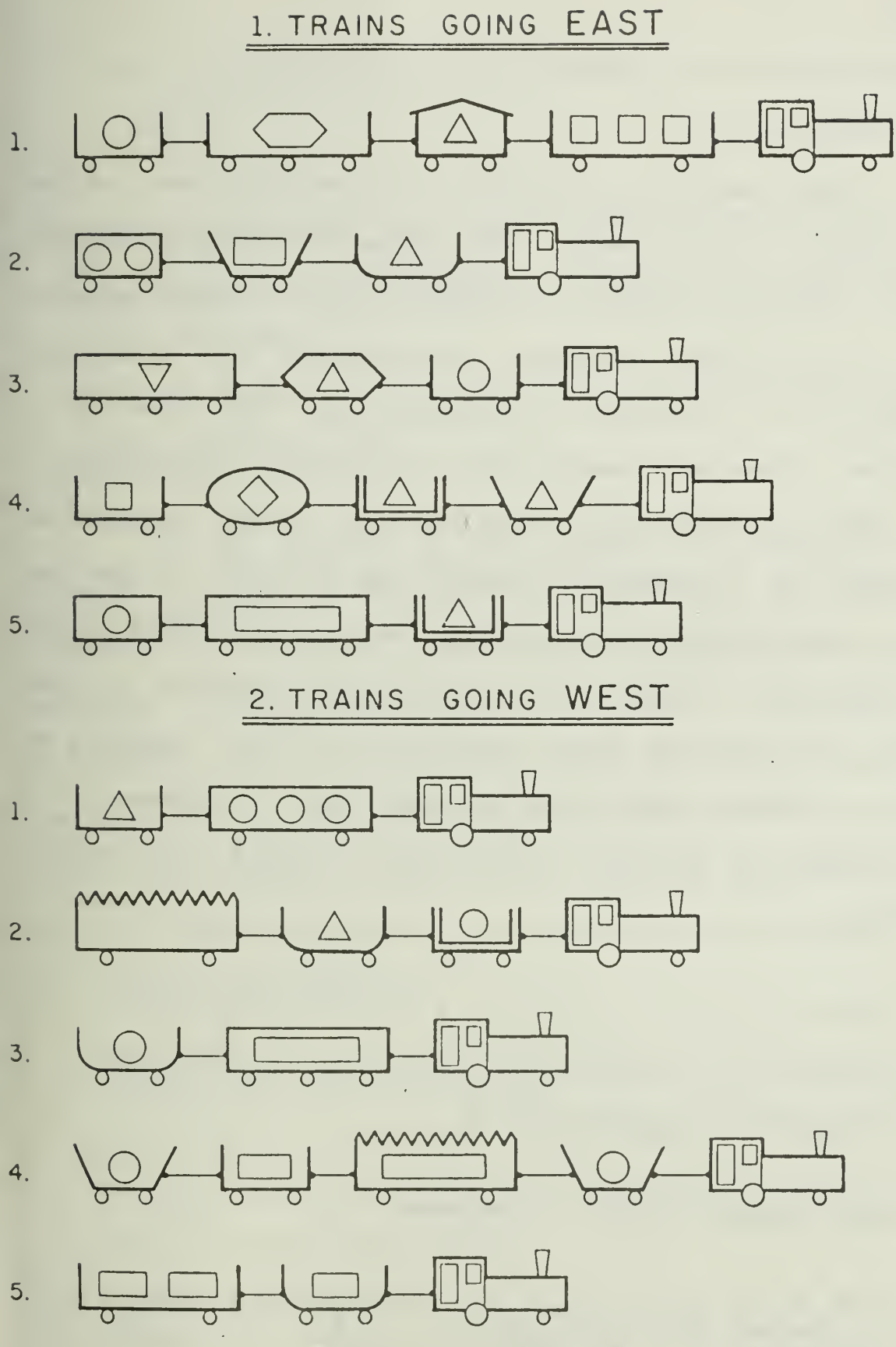
[$PT(ln=1]=2][loc($car_1$) =3][cshape($car_1$) =10]=>[d=2].

(If there are two long cars and either all cars have 2 wheels or the third car has an open top, then the train is going west.)

[#PT(ln=1) =2][cshape(car_1) =1,2] v

[loc(car_1) =2][cshape(car_1) =6]=>[d=2]

(If there are exactly 2 long cars, one of which has shape open trapazoid or U-shape or car number 2 is a closed rectangle, then the train is going west.)



Trains for Example EX 3

Figure 6.3

6.4 Textures (EX 4)

This example is included to give some idea of the limitations of the program. The figure 6.4 (from [Michalski 1972]) contains examples of figures with 9 boxes numbered from 1 to 9 from the center to the right and around the outside in a clockwise direction each with one of 4 textures. There are several ways to represent the figures. The first representation takes advantage of the ordering of arguments in a function to specify the location of each box (i.e, the third box corresponds to the third argument of the predicate p). Since there is no apparant structure to each object, the program relies heavily on the AQ7 procedure to find a simple description of each set of objects. A description of the first object in the set T_1 the results follow:

$$[p(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)][tx(x_1)=0][tx(x_2)=3]$$
$$[tx(x_3)=1][tx(x_4)=3][tx(x_5)=1][tx(x_6)=3][tx(x_7)=0]$$
$$[tx(x_8)=2][tx(x_9)=0] \Rightarrow [d=1]$$

Resulting decision rules: (15 seconds)

$$[p(x_1,x_2,x_3,x_4,x_3,x_6,x_7,x_8,x_9)][tx(x_3)=0,2,3][tx(x_6)=1,2]$$
$$[tx(x_7)=0,1] \vee$$

$[p(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)]\ [tx(x_2)=0,3]\ [tx(x_5)=1,2]$
$\Rightarrow [d=1]$

$[p(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)]\ [tx(x_2)=1]\ [tx(x_7)=2]\ \vee$
$[p(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)]\ [tx(x_2)=1,2]\ [tx(x_3)=1,2]$
$[tx(x_4)=0,1,2]\ \vee$
$[p(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9)]\ [tx(x_2)=0,3]\ [tx(x_3)=0,2,3]$
$[tx(x_5)=0,3] \Rightarrow [d=2]$

If the program is now allowed to generalize over location as well as texture, then the results are much more difficult to attain. The above representation may be modified by specifying that the order of arguments is irrelevant to the predicate p (using a (.) between arguments instead of a (,)) and specifying the location of a box in the object by assigning a value to the variable representing the box (assigning a subrange $x_i=2$, for example). The description of an object then appears:

$[p(x_1.x_2.x_3.x_4.x_5.x_6.x_7.x_8.x_9)]\ [x_1=1]\ [x_2=2]\ [x_3=3]\ [x_4=4]$
$[x_5=5]\ [x_6=6]\ [x_7=7]\ [x_8=8]\ [x_9=9]\ [tx(x_1)=0]\ [tx(x_2)=3]$
$[tx(x_3)=1]\ [tx(x_4)=3]\ [tx(x_5)=1]\ [tx(x_6)=3]\ [tx(x_7)=0]$
$[tx(x_8)=2]\ [tx(x_9)=0] \rightarrow [d=1].$

The program is allowed to generalize over values of both

$tx(x_i)$ and x_i. The features of the graph matching algorithm which before lead to efficient comparisons between objects are no longer useful (i.e., the fact that all nodes are labelled and that many edges are labelled). Since the c-structure of a formula of the form

$$[x_1=3][tx(x_1)=2][p(x_1,x_2)][x_2=1][tx(x_2)=1]$$

is

$$[x_1=*][tx(x_1)=*][p(x_1,x_2)][x_2=*][tx(x_2)=*]$$

there are very many occurrences of such a c-structure in each object resulting in a large number of VL_1 events supplied to the AQ7 procedure (200 events in each set with from 6 to 8 variables). The program was run with the value 1 for each parameter but unfotunately the program ran for over 1 minute with no results. The input set of decision rules was then cut to only the first 5 objects in set T_1 and the last five objects in set T_2 and the program was run again with this reduced set of objects. The results of the program execution using this formulation are: (1 minute)

$$[p(x_1,x_2)][x_1=1][x_2=9][tx(x_1)=2][tx(x_2)=1] \vee$$
$$[p(x_1,x_2)][x_1=1][x_2=5.9][tx(x_1)=0.3][tx(x_2)=1.2.3] \vee$$
$$[x_1=3][tx(x_1)=3] \Rightarrow [d=1]$$

$[p(x_1 \cdot x_2)][x_1=1][x_2=4][tx(x_1)=3][tx(x_2)=3] \vee$

$[p(x_1 \cdot x_2)][x_1=1][x_2=9][tx(x_1)=2][tx(x_2)=0] \vee$

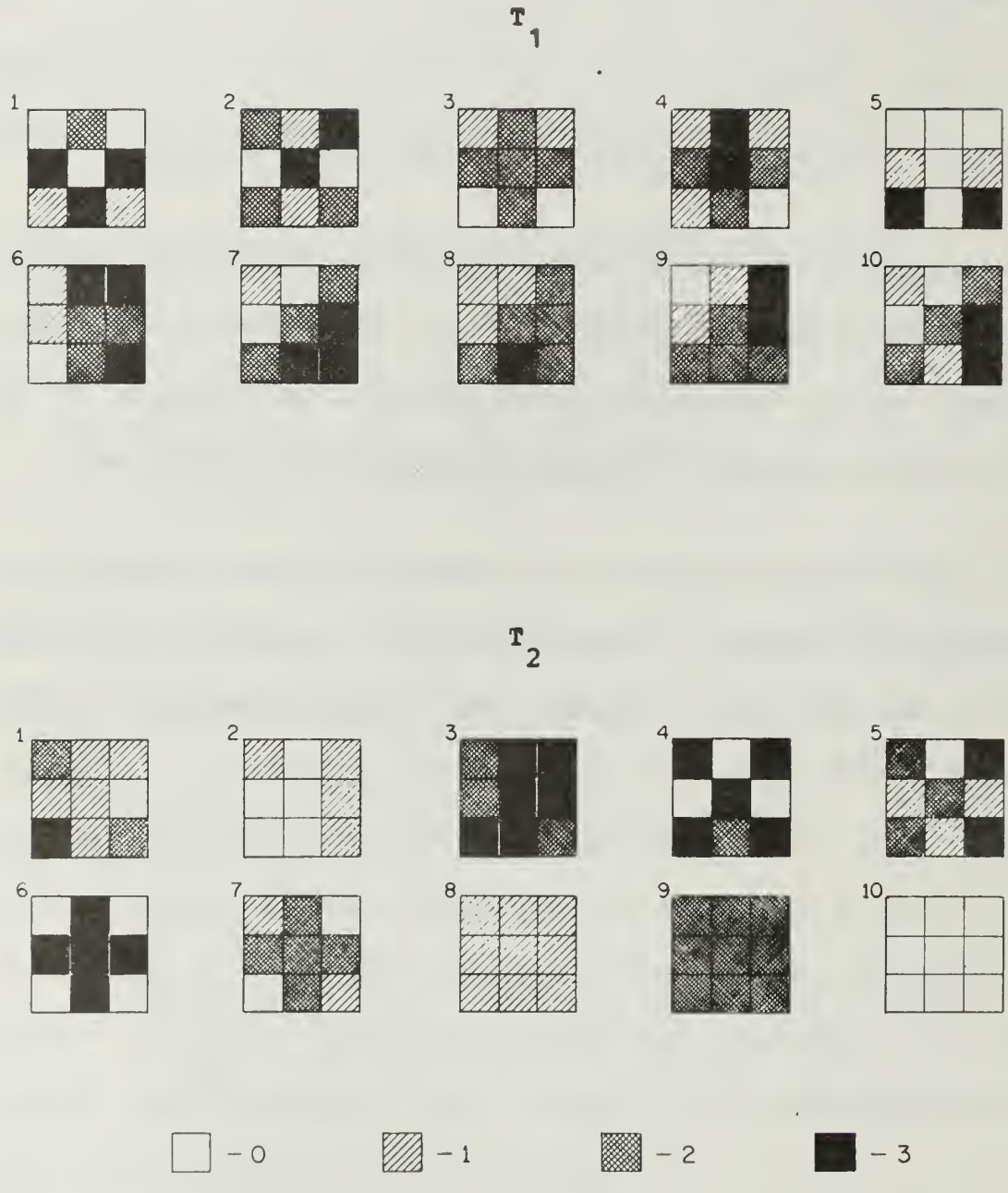
$[x_1=1][tx(x_1)=1] \vee$

$[x_1=9][tx(x_1)=2] \vee$

$[p(x_1 \cdot x_2)][x_1=1][x_2=5][tx(x_1)=0][tx(x_2)=0] \Rightarrow [d=2]$

(Presumably, if the parameters are increased, the results would be much more interesting but this appears to be the limit of the program's applicability with regard to the exponential nature of the graph isomorphism algorithm.)

The time required to compute these results is significantly greater than the time required when the location of each box is known (15 seconds compared to one minute).



Texture Figures for Example EX 4

Figure 6.4

7. Current Limitations and Possible Extentions

In the preceeding chapters, a formal methodology has been presented for solving a wide variety of inductive tasks. It differs from previous work in it's formal basis, flexible optimality criteria and domain definitions, ability to add new descriptors to existing descriptions, and apparent extensibility. Presented below are a few limitations of the current implementation. Some of the limitations have plausible solutions in light of some applications by other authors in specific problem areas. Some extentions may be added to the program with the current state of the art and are omitted from the current implementation due to the higher priority placed on evaluating the existing features. Some features require algorithms which are not yet well understood (expecially those which involve the modification of hypotheses in light of new information and the use of multi-valued truth domains). Extensions are listed in the author's estimate of increasing difficulty.

1. The restriction which requires generalizations to have weakly connected graph structure representations has the advantage that the search space is limited and resulting descriptions contain selectors which

are related in some meaningful way. This constraint should be applied at the option of the user.

2. The graph structure used in the current implementation has not been fully analyzed. It appears that unary functions represent a special type of description of parts of objects and are well suited to generalizations using a VL_1 system. Functions and predicates of higher order represent relationships between parts and are more amenable to generalization using a VL_2 approach (i.e.,, 'growing' consistent generalizations). These two roles of functions and predicates have not been investigated.

3. The program has a potential for application to more abstract objects (e.g. patient records used by the MYCIN system). Up until now, only simple physical objects have been used as examples.

4. There should be a facility to add new user defined functions to a rule of the type:

$$V \quad \models \quad V[SUM\_cost=3]$$

where V is $[cost(x_1)=2][cost(x_2)=1]$

5. The tree structure domain may be extended to a generalization structure as presented by Michalski [Michalski 75].

6. Only the operators (v and &) are currently understood by the program. The VL_2 system contains a very rich set of operators, one of the most important being the exception operator. In the current implementation, there is no generalization of decision D=1 in the set of rules:

$$[p(x_1,x_2)] \Rightarrow [D=1]$$
$$[p(x_1,x_2)][s(x_1)=1] \Rightarrow [D=2]$$

7. Decision rules which are used heavily in production systems contain condition and decision parts which share arguments. These should be handled by the program in some way. e.g.

$$[p(x_1,x_2)] \Rightarrow [q(x_1,x_2)]$$

8. Descriptive generalizations only contain one conjunctive expression. In some circumstances, it is desirable to produce a disjunctive, descriptive description (see Yuen for a program UNICLASS which

generates descriptive descriptions for VL_1 expressions).

9. The order in which objects are selected from the set F1 seems to be significant. The program ESEL whose algorithm is described in Michalski [Michalski 75] gives a method for selecting VL_1 type events.

10. Universal quantifiers have been considered only in a global sense since a simple universally quantified expression is less general than an existentially quantified expression in the condition part of a rule. A universal or distinct universal quantifier could be added to the program.

11. Only the distinct existential quantifier is included in the program. This decision was made in order to limit the search space as well as to simplify the user input and the resulting expressions. Hayes-Roth [Hayes-Roth 76] has pointed out the utility of a 'many to one' mapping instead of the current one to one map.

12. No facility is included in the current implementation to accomodate initial hypotheses in order to form a

new consistent, complete generalization in light of new information. Larson [Larson 76] and Hedrick [Hedrick 74] give some approaches to this problem using other systems. For example, given an hypothesis:

$$[p(x_1,x_2)][s(x_1)=1] \Rightarrow [D=1]$$

and the new information that

$$[p(x_1,x_2)][s(x_2)=2] \Rightarrow [D=2],$$

the program should generate a new rule which is consistent (e.g.,

$$[p(x_1,x_2)][s(x_2)\neg=2] \Rightarrow [D=1]$$

13. Generalizations of subsets of parts of an object should be usable by the program to form new, higher level generaliztions in terms of the these generalized concepts (e.g. Winston [Winston 70] generates the description of an arch and then uses this description to describe a sequence of arches).

14. Include a multi-valued truth domain to learn from data of differing degrees of certainty. Very little

is done in this area; one of the problems being that a deductive system using multi-valued truth values is not well understood.

LIST OF REFERENCES

Banerji, R. 1975, "Learning with Structural Descriptions." working paper Temple University, Philadelphia, Penn.

Bongard, M. 1970, Pattern Recognition., translated by Theodore Chevon, Spartan books, New York, N. Y.

Buchanan, B.G., Sutherland, G.L., and Feigenbaum, E.A. 1969, "Heuristic DENDRAL: a Program for Generating Explanatory Hypotheses in Organic Chemistry", in Machine Intelligence 4. (B. Meltzer and D. Michie eds), Edinburgh University Press.

Buchanan, B.G., Sutherland, G.L., and Feigenbaum, E.A. 1972, "Heuristic Theory Formation, Data Interpretation, and Rule Formation." Machine Intelligence 7. (B. Meltzer and D. Michie eds), John Wiley & Sons.

Chilausky R., Jacobsen B., Michalski R.S. 1976, "An application of Variable Valued Logic to Inductive Learning of Plant Disease Diagnostic Rules", Proceedings of the Sixth Annual Symposium on Multiple Valued Logic. Logan, Utah.

Croft, J.A. 1971, "A comparative Study of Mathematical Methods for Diagnosing Disease." Ph.D dissertation, Northwestern University Evanston, Illinois.

Hayes-Roth, F. and McDermott, J. 1976, "Knowledge Acquisition from Structural Descriptions." Department of Computer Science, Carnegie Mellon University.

Hedrick C.L. 1974, "A Computer Program to Learn Production Systems Using a Semantic Net." Department of Computer Science, Carnegie Mellon University, Pittsburgh, Penn.

Hunt, E.B. 1966, Experiments in Induction, Academic Press.

Kochen, M. 1974, "Representation and Algorithms for Cognitive Learning." Artificial Intelligence 5.

Larson J., Michalski R.S., "AQVAL/1-AQ7 User's Guide and Program Description." Department of Computer Science report number 731, University of Illinois.

Larson J. 1976, "A Multi-Step Formation of Variable Valued Logic Hypotheses." Proceedings of the Sixth International Symposium on Multiple Valued Logic, Logan, Utah.

Larson J., Michalski R.S. 1977, "Inductive Inference of VL Decision Rules." Workshop on Pattern Directed Inference Systems, Hawaii.

Michalski R.S. 1972, "A Variable-Valued Logic System as Applied to Picture Description and Recognition." Graphic Languages, proceedings of the IFIIP Working Conference on Graphic Languages. Vancouver Canada.,

Michalski R.S. 1973, "AQVAL/1 AQ7 Computer Implementation of a Variable Valued Logic System and the Application to Pattern Recognition." Proceedings of the First International Joint Conference on Pattern Recognition. Washington, D.C.

Michalski R.S. 1974a, "Variable-Valued Logic: System VL_1." Proceedings of Fourth International Symposium on Mutiple-Valued Logic. West Virginia University, Morgantown, Virginia.

Michalski R.S. 1974b, "Learning by Inductive Inference." NATO Advanced Study Institute on Computer Oriented Learning Process." France.

Michalski R.S. 1974c, "Problems of Designing an Inferential

Medical Consulting System." First Illinois Conference on Medical Information Systems University of Illinois Urbana, Illinois.

Michalski R.S. 1975, "On the Selection of Representative Samples From Large Relational Tables for Inductive Inference." Department of Information Engineering University of Illinois at Chicago Circle, Chicago, Illinois.

Michalski R.S. 1977, "Toward Computer-aided Induction: a brief review of currently implemented AQVAL programs". 5th International Conference on Artificial Intelligence, Cambridge, Mass.

Morgan C.G. 1972, "Inductive Resolution." Master's thesis, Department of Computer Science, Edmonton Alberta.

Pople, H., Werner G. 1972, "An Information Processing Approach to Theory Formation in Biomedical Research." AIFIPS Conference Proceedings. Vol 40.

Rychener, M.D. 1976, "Production Systems as a Programming Language for Artificial Intelligence Applications." Ph.d dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Penn.

Schubert K.L. 1976, "Extending the Expressive Power of Semantic Networks." Artificial Intelligence 7.

Shortliffe, E.H. 1974, "A Rule Based Computer Program for Advising Physicians Regarding Antimicrobial Therapy Selection." Ph.D dissertation." Stanford Artificial Intelligence Laboratory, Memo AIM-251.

Tarjan R. 1972, "Depth-First Search and Linear Graph Algorthms." SIAM Journal on Computers. Vol 1, No 2.

Vere S.A. 1975, "Induction of Concepts in the Predicate Calculus." Proceedings of the Fourth International Joint Conference on Artificial Intelligence.

Waterman D.H. 1970, "Generalization of Learning Techniques for Automating the Learning of Heurisics." Artificial Intelligence 1.

Waterman D.H. 1974, "Adaptive Production Systems." working paper #385, Department of Psychology, Carnegie Mellon University, Pittsburgh, Penn.

Waterman D.A. 1975, "Serial Pattern Acquisition: A Production System Approach." working paper #286,

Department of Psychology, Carnegie Mellon University, Pittsburgh, Penn.

Winston, D.H. 1970, "Learning Structural Descriptions from Examples." AI-TR-76, Cambridge; MIT, Artificial Intelligence Laboratory.

Yuen H., "Uniclass Synthesis: User's Guide" internal report. Department of Coomputer Science, University of Illinois. Urbana, Illinois.

APPENDIX A

This appendix contains a trace of the program execution through the solution of EX1 (section 6.1) covering the set O_1 with c-formulas with the EXTNTY type predicates added to each input rule. Most of the output is self explanatory (with reference to chapter 6 for the definition of some terms), with the possible exception of the output of rules. Rules are printed with a rule number (a unique number assigned by the program to each new graph data structure which is needed). The event set which the rule is intended to cover (i.e., the decision value in the decision D=i) is given after the rule number the cost function indices used in trimming (in parentheses after the word CCSTS) is given. Following this is a list of the actual costs of the c-formula). Comments are interspersed with the output to clarify specific points. (Computer output is given in capital letters).

Parameters:

AQPARMSS

AQMAXSTAR = 7 LQST NCONSIST = 10

AQCRIT	AQTOLERANCE
-1	0
2	0
4	0

NBR OF CRIT: AQNF = 2

VLPARMS

ALTER = 4 VLMAXSTAR = 4

VLCRIT	VLTOLERANCE
3	0.30
-1	0
2	0

VLNF = 3

```
NEW FNCTNS:       METATRIM = 0

NOW COVERING EVENT
RULE         5 EVENT SETS:  1 COSTS( 3-1 2)     0     0     0
[ONTOP(P1,P2)][ONTOP(P2,P3)][SIZE(P1)= 1][SIZE(P2)= 1]
    [SIZE(P3)= 2][SHAPE(P1)= 0][SHAPE(P2)= 2][SHAPE(P3)= 5]
    [TX(P1)= 0][TX(P2)= 1][TX(P3)= 0][MST-ONTOP(P1)]
    [LST-ONTOP(P3)]
```

A c-formula has been selected from the set O_1 which is represented by rule number 5.

```
THE FOLLOWING FORMULAS ARE IN THE UNTRIMMED STAR
RULE        22 EVENT SETS:  1 COSTS( 3-1 2)     6    -3     1
[LST-ONTOP(P1)]
RULE        21 EVENT SETS:  1 COSTS( 3-1 2)     6    -3     1
[MST-ONTOP(P1)]
RULE        20 EVENT SETS:  1 COSTS( 3-1 2)     3    -3     1
[TX(P1)= 0]
RULE        19 EVENT SETS:  1 COSTS( 3-1 2)     6    -3     1
[TX(P1)= 1]
RULE        18 EVENT SETS:  1 COSTS( 3-1 2)     3    -3     1
[TX(P1)= 0]
RULE        17 EVENT SETS:  1 COSTS( 3-1 2)     3    -1     1
[SHAPE(P1)= 5]
RULE        16 EVENT SETS:  1 COSTS( 3-1 2)     2    -2     1
[SHAPE(P1)= 2]
RULE        15 EVENT SETS:  1 COSTS( 3-1 2)     2    -1     1
[SHAPE(P1)= 0]
RULE        14 EVENT SETS:  1 COSTS( 3-1 2)     4    -2     1
[SIZE(P1)= 2]
RULE        13 EVENT SETS:  1 COSTS( 3-1 2)     5    -3     1
[SIZE(P1)= 1]
RULE        12 EVENT SETS:  1 COSTS( 3-1 2)     5    -3     1
[SIZE(P1)= 1]
```

All selectors of the event (condition of rule 5) are place in the intial partial star. Since VLMAXSTAR has the value 4, the above list is trimmed to a smaller list of the 4 best c-formulas in the partial star.

```
THE FOLLOWING FORMULAS REMAIN AFTER TRIMMING
RULE        16 EVENT SETS:  1 COSTS( 3-1 2)    2   -2    1
[SHAPE(P1)= 2]
RULE        14 EVENT SETS:  1 COSTS( 3-1 2)    4   -2    1
[SIZE(P1)= 2]
RULE        12 EVENT SETS:  1 COSTS( 3-1 2)    5   -3    1
[SIZE(P1)= 1]
RULE        18 EVENT SETS:  1 COSTS( 3-1 2)    3   -3    1
[TX(P1)= 0]
```

The next partial star is formed by adding one selector to all c-formulas in the previous partial star forming a list of c-formulas which have two selectors.

```
THE FOLLOWING FORMULAS ARE IN THE UNTRIMMED STAR
RULE        38 EVENT SETS:  1 COSTS( 3-1 2)    1   -3    2
[TX(P1)= 0][MST-ONTOP(P1)]
RULE        37 EVENT SETS:  1 COSTS( 3-1 2)    1   -1    2
[SHAPE(P1)= 0][TX(P1)= 0]
RULE        36 EVENT SETS:  1 COSTS( 3-1 2)    2   -3    2
[SIZE(P1)= 1][TX(P1)= 0]
RULE        35 EVENT SETS:  1 COSTS( 3-1 2)    3   -3    2
[ONTOP(P1,P2)][TX(P1)= 0]
RULE        34 EVENT SETS:  1 COSTS( 3-1 2)    3   -3    2
[SIZE(P1)= 1][MST-ONTOP(P1)]
RULE        33 EVENT SETS:  1 COSTS( 3-1 2)    2   -3    2
```

```
[SIZE(P1)= 1][TX(P1)= 0]
RULE        32 EVENT SETS:  1 COSTS( 3-1 2)    2   -1    2
[SIZE(P1)= 1][SHAPE(P1)= 0]
RULE        31 EVENT SETS:  1 COSTS( 3-1 2)    4   -3    2
[ONTOP(P1,P2) ][SIZE(P1)= 1]
RULE        30 EVENT SETS:  1 COSTS( 3-1 2)    3   -2    2
[SIZE(P1)= 2][LST-ONTOP(P1) ]
RULE        29 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    2
[SIZE(P1)= 2][TX(P1)= 0]
RULE        28 EVENT SETS:  1 COSTS( 3-1 2)    3   -1    2
[SIZE(P1)= 2][SHAPE(P1)= 5]
RULE        27 EVENT SETS:  1 COSTS( 3-1 2)    3   -2    2
[ONTOP(P1,P2) ][SIZE(P2)= 2]
RULE        26 EVENT SETS:  1 COSTS( 3-1 2)    1   -1    2
[SHAPE(P1)= 2][TX(P1)= 1]
RULE        25 EVENT SETS:  1 COSTS( 3-1 2)    2   -2    2
[SIZE(P1)= 1][SHAPE(P1)= 2]
RULE        24 EVENT SETS:  1 COSTS( 3-1 2)    2   -1    2
[ONTOP(P1,P2) ][SHAPE(P1)= 2]
RULE        23 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    2
[ONTOP(P1,P2) ][SHAPE(P2)= 2]
```

At most 4 new selectors were added to each c-formula of the previous partial star (ALTER=4). This partial star is now trimmed to the 4 best elements according to the optimality criterion.

```
THE FOLLOWING FORMULAS REMAIN AFTER TRIMMING
RULE        25 EVENT SETS:  1 COSTS( 3-1 2)    2   -2    2
[SIZE(P1)= 1][SHAPE(P1)= 2]
RULE        23 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    2
[ONTOP(P1,P2) ][SHAPE(P2)= 2]
RULE        29 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    2
[SIZE(P1)= 2][TX(P1)= 0]
```

```
RULE        38 EVENT SETS:  1 COSTS( 3-1 2)    1   -3    2
[TX(P1)= 0][MST-ONTOP(P1)]
THE FOLLOWING FORMULAS ARE IN THE UNTRIMMED STAR
RULE        51 EVENT SETS:  1 COSTS( 3-1 2)    0   -1    3
[SHAPE(P1)= 0][TX(P1)= 0][MST-ONTOP(P1)]
RULE        50 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    3
[SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
RULE        49 EVENT SETS:  1 COSTS( 3-1 2)    1   -3    3
[ONTOP(P1,P2)][TX(P1)= 0][MST-ONTOP(P1)]
RULE        48 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[SIZE(P1)= 2][TX(P1)= 0][LST-ONTOP(P1)]
RULE        47 EVENT SETS:  1 COSTS( 3-1 2)    1   -1    3
[SIZE(P1)= 2][SHAPE(P1)= 5][TX(P1)= 0]
RULE        46 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P2)= 0]
RULE        45 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SHAPE(P2)= 2][TX(P1)= 0]
RULE        44 EVENT SETS:  1 COSTS( 3-1 2)    0   -1    3
[ONTOP(P1,P2)][SHAPE(P1)= 0][SHAPE(P2)= 2]
RULE        43 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    3
[ONTOP(P1,P2)][SIZE(P2)= 1][SHAPE(P2)= 2]
RULE        42 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    3
[ONTOP(P1,P2)][SIZE(P1)= 1][SHAPE(P2)= 2]
RULE        41 EVENT SETS:  1 COSTS( 3-1 2)    1   -1    3
[SIZE(P1)= 1][SHAPE(P1)= 2][TX(P1)= 1]
RULE        40 EVENT SETS:  1 COSTS( 3-1 2)    2   -1    3
[ONTOP(P1,P2)][SIZE(P1)= 1][SHAPE(P1)= 2]
RULE        39 EVENT SETS:  1 COSTS( 3-1 2)    1   -2    3
[ONTOP(P1,P2)][SIZE(P2)= 1][SHAPE(P2)= 2]
THE FOLLOWING FORMULAS REMAIN AFTER TRIMMING
RULE        56 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SHAPE(P2)= 2][TX(P1)= 0]
RULE        55 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P2)= 0]
RULE        54 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[SIZE(P1)= 2][TX(P1)= 0][LST-ONTOP(P1)]
```

```
RULE        53 EVENT SETS:   1 COSTS( 3-1 2)     0    -3     3
[SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
```

Since each selected c-formula is consistent, all c-formulas are placed in the list MQ. The program now looks for more descriptive generalizations to supply a larger set of variables to the AQ7 procedure.

```
THE FOLLOWING FORMULAS ARE IN THE UNTRIMMED STAR
RULE        69 EVENT SETS:   1 COSTS( 3-1 2)     0    -1     4
[SIZE(P1)= 1][SHAPE(P1)= 0][TX(P1)= 0][MST-ONTOP(P1)]
RULE        68 EVENT SETS:   1 COSTS( 3-1 2)     0    -3     4
[ONTOP(P1,P2)][SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
RULE        67 EVENT SETS:   1 COSTS( 3-1 2)     0    -1     4
[SIZE(P1)= 2][SHAPE(P1)= 5][TX(P1)= 0][LST-ONTOP(P1)]
RULE        66 EVENT SETS:   1 COSTS( 3-1 2)     0    -2     4
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P2)= 0][LST-ONTOP(P2)]
RULE        65 EVENT SETS:   1 COSTS( 3-1 2)     0    -2     4
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P1)= 1][TX(P2)= 0]
RULE        64 EVENT SETS:   1 COSTS( 3-1 2)     0    -1     4
[ONTOP(P1,P2)][SIZE(P2)= 2][SHAPE(P2)= 5][TX(P2)= 0]
RULE        63 EVENT SETS:   1 COSTS( 3-1 2)     0    -1     4
[ONTOP(P1,P2)][SIZE(P2)= 2][SHAPE(P1)= 2][TX(P2)= 0]
RULE        62 EVENT SETS:   1 COSTS( 3-1 2)     0    -2     4
[ONTOP(P1,P2)][SIZE(P1)= 1][SIZE(P2)= 2][TX(P2)= 0]
RULE        61 EVENT SETS:   1 COSTS( 3-1 2)     0    -1     4
[ONTOP(P1,P2)][SHAPE(P2)= 2][TX(P1)= 0][TX(P2)= 1]
RULE        60 EVENT SETS:   1 COSTS( 3-1 2)     0    -1     4
[ONTOP(P1,P2)][SHAPE(P1)= 0][SHAPE(P2)= 2][TX(P1)= 0]
RULE        59 EVENT SETS:   1 COSTS( 3-1 2)     0    -2     4
[ONTOP(P1,P2)][SIZE(P2)= 1][SHAPE(P2)= 2][TX(P1)= 0]
RULE        58 EVENT SETS:   1 COSTS( 3-1 2)     0    -2     4
[ONTOP(P1,P2)][SIZE(P1)= 1][SHAPE(P2)= 2][TX(P1)= 0]
```

At least 10 consistent formulas have be generated and placed in MQ. The AQ procedure is then applied to each one of the consistent formulas (NCONSIST=10). Only the trace of one such application is given below.

```
THE CONSISTENT FORMULAS:
BEFORE AQ:
RULE       58 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    4
[ONTOP(P1,P2) ][SIZE(P1)= 1 ][SHAPE(P2)= 2 ][TX(P1)= 0 ]
THE C-FORMULA STRUCTURE IS:
RULE       58 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    4
[ONTOP(P1,P2) ][SIZE(P1)= * ][SHAPE(P2)= * ][TX(P1)= * ]
THERE ARE   6 VL1 TYPE VARIABLES X1,X2,...,X 6
VARIABLES ARE ASSOCIATED WITH NODES IN THE C-FORMULA AS FOLLOWS:
                NODE                 VARIABLE
                ONTOP                X1
                P1                   X2
                P2                   X3
                SIZE                 X4
                SHAPE                X5
                TX                   X6
AQ IS APPLIED TO THE FOLLOWING INPUT CPXS/EVENTS
                                     SET 1
[X1=  1][X4=  1][X5=  2][X6=  0]
[X1=  1][X4=  1][X5=  4][X6=  1]
[X1=  1][X4=  1][X5=  1][X6=  0]
[X1=  1][X4=  1][X5=  2][X6=  0]
                                     SET 2
[X1=  1][X4=  2][X5=  1][X6=  0]
[X1=  1][X4=  1][X5=  4][X6=  0]
[X1=  1][X4=  2][X5=  0][X6=  1]
[X1=  1][X4=  1][X5=  1][X6=  0]
[X1=  1][X4=  1][X5=  2][X6=  1]
[X1=  1][X4=  1][X5=  5][X6=  1]
```

```
[X1=  1][X4=  1][X5=  1][X6=  1]
[X1=  1][X4=  2][X5=  6][X6=  1]
[X1=  1][X4=  0][X5=  5][X6=  1]
THE RESULTING COMPLEX FROM THIS PASS IS:
[X4=  1][X5=  0  2  3  7  9][X6=  0]
AFTER AQ:
RULE       58 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    4
[ONTOP(P1,P2)][SIZE(P1)= 1][SHAPE(P2)= 9][TX(P1)= 0]
```

The results from the application of the AQ7 procedure to each consistent formula are given below. These are the consistent alternative solutions (note that only those with a cost of -3 for cost function -1 are complete).

```
THE FOLLOWING ARE ALTERNATIVECONSISTENT GENERALIZATIONS
RULE       58 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    4
[ONTOP(P1,P2)][SIZE(P1)= 1][SHAPE(P2)= 9][TX(P1)= 0]
RULE       59 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SIZE(P2)= *][SHAPE(P2)= 2][TX(P1)= 0]
RULE       60 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SHAPE(P1)= 9][SHAPE(P2)= 9][TX(P1)= *]
RULE       61 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SHAPE(P2)= 2][TX(P1)= 0][TX(P2)= *]
RULE       62 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    4
[ONTOP(P1,P2)][SIZE(P1)= 1][SIZE(P2)= 2][TX(P2)= 0]
RULE       63 EVENT SETS:  1 COSTS( 3-1 2)    0   -1    3
[ONTOP(P1,P2)][SIZE(P2)= 2][SHAPE(P1)= 2][TX(P2)= *]
RULE       64 EVENT SETS:  1 COSTS( 3-1 2)    0   -1    4
[ONTOP(P1,P2)][SIZE(P2)= 2][SHAPE(P2)= 8][TX(P2)= 0]
RULE       65 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    4
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P1)= 1][TX(P2)= 0]
RULE       66 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P2)= 0][LST-ONTOP(P2)]
RULE       67 EVENT SETS:  1 COSTS( 3-1 2)    0   -1    3
```

```
[SIZE(P1)= 2][SHAPE(P1)= 8][TX(P1)= 0][LST-ONTOP(P1)]
RULE        68 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    3
[ONTOP(P1,P2)][SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
RULE        69 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    3
[SIZE(P1)= 1][SHAPE(P1)= 9][TX(P1)= 0][MST-ONTOP(P1)]
RULE        44 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SHAPE(P1)= 9][SHAPE(P2)= 9]
RULE        45 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SHAPE(P2)= 2][TX(P1)= 0]
RULE        46 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    3
[ONTOP(P1,P2)][SIZE(P2)= 2][TX(P2)= 0]
RULE        48 EVENT SETS:  1 COSTS( 3-1 2)    0   -2    2
[SIZE(P1)= 2][TX(P1)= 0][LST-ONTOP(P1)]
RULE        50 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    2
[SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
RULE        51 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    2
[SHAPE(P1)= 9][TX(P1)= 0][MST-ONTOP(P1)]
THE SELECTED MQ FORMULA IS:
RULE        50 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    2
[SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
```

The solution to the problem is the following rule. Note however that there were four distinct solutions generated (rules 68, 69, 50, and 51).

```
THE FOLLOWING FORMULAS COVER SET          1
THIS RULE COVERS          3 NEW RULES
RULE        50 EVENT SETS:  1 COSTS( 3-1 2)    0   -3    2
[SIZE(P1)= 1][TX(P1)= 0][MST-ONTOP(P1)]
```

APPENDIX B

This appendix contains a very brief description of the AQ7 procedure implemented in the program INDUCE_1. The reader is referred to [Larson et.al. 75] for further details. A set of variables $(x_1, x_2, \ldots, x_n)$ and a domain definition including size and structure (nominal, interval or tree-structured) is given. The space (event space) defined by

$$E = D(x_1) \times D(x_2) \times \ldots \times D(x_n)$$

contains a set of points (events) where each point may be specified by supplying a list of values for the variables $x_1, x_2, \ldots, x_n$. A complex is the set of points in the event space which satisfy a particular VL_1 expression of the form

$$[x_1 = R_1][x_2 = R_2] \ldots [x_n = R_n]$$

where R_i is a representation for a subset of the domain $D(x_i)$ or the symbol (*) (if R_i=*, then the selector may be removed from the expression). The procedure accepts two sets of VL_1 expressions (L and L') representing disjoint complexes and produces a near optimal generalization of the set of expressions in L with regard to a set of optimality criteria. Certain other parameters control the generalization process.

Let $L=[l_1,l_2,..,l_m]$ and $L'=[l'_1,l'_2,....,l'_{m'}]$ (these two sets are denoted F1 and F0 respectively in other references; the above notation was selected to avoid confusion with sets of c-formulas). The program selects one element (e_0) from L and forms a star about this element (a star about e_0 is a set of complexes each of which contains e_0 but does not intersect with any element of L' and is nearly maximal under inclusion). One element (l_q) is selected from the star using the optimality criterion and placed in a set of output complexes; all other elements of L which are covered by this l_q are removed from L and the process is repeated with a new e_0 until the set L is exhausted.

A star is generated by forming a sequence of elementary stars and partial stars, one for each element of L'. An elementary star ($ES(e_0,l'_i)$ or ES_i) is a set of complexes which covers e_0 does not intersect with l'_i, and is maximal under inclusion and under domain structure constraints. A partial star ($P_i(e_0)$ or P_i) is a set of complexes which contain e_0 but do not intersect with any l'_j, $j<=i$ (note that $P_{m'}$ is in fact a star). To generate an elementary star ES_i, the extension against rule (see chapter 3) is applied to each selector of e_0 in the context of l'_i. The result is a set of selectors, each one corresponding to one selector of e_0. To form a partial star P_{i+1} from a

partial star P_i, each element of P_i is multiplied by each element of ES_{i+1} (i.e., the set of complexes in P_i and ES_i may be viewed as a logical sum of products of selectors; the multiplication is then the normal result of expanding a product of sums in the VL_1 system).

The partial star P_0 is initialized to be the entire event space E. As each partial star P_i is generated, absorption laws are applied to discard any complex of P_i which is contained in another complex of P_i. The partial star is then trimmed to AQMAXSTAR number of elements using a procedure identical to that in section 5.4. The final partial star ($P_{m'}$) is trimmed with an AQMAXSTAR value of 1 to produce l_q. If a parameter LQST is set (has the value TRUE), then l_q is stripped down to an expression with the following properties:

1) the stripped l_q contains the same variables as the original expression,
2) the stripped l_q covers the same elements of L as the original,
3) the reference of each selector of l_q contains the fewest elements of all complexes satisfying 1 and 2 above under domain structure constraints (i.e., interval variables must have a range of values in the reference).

If the set L' is null, then (1) above is replaced with:

1) the stripped l_q contains all variables $x_1, x_2, \ldots, x_n$.

The latter condition occurs when generating descriptive descriptions covering only one set of complexes (see sections 6.1,6.2). Stripping is done by finding the disjunction of all complexes in L covered by l_q and then adjusting the reference of each selector in the sum to conform to the domain structure (i.e., for interval domains, this involves filling the gaps to while form an interval, while for tree structured domains, this involves finding the lowest level generalization of all elements in the reference).

A simple example may clarify the procedure. Given 3 variables with domains:

Variable	Structure	Values
x_1	nominal	[0:2]
x_2	interval	[0:3]
x_3	tree-struc	[0:6]
	0,1,2=>5	
	3,4=>6	

and input complexes and parameters:

L: l_1 [x_1=0] [x_2=1] [x_3=2]

l_2 [x_1=1] [x_2=1] [x_3=0]

l_3 [x_1=1] [x_2=2] [x_3=0]

L': l'_1 [x_1=1] [x_2=2] [x_3=3]

l'_2 [x_1=0] [x_2=3] [x_3=1]

l'_3 [x_1=3] [x_2=2] [x_3=4]

AQMAXSTAR = 2 LQST = TRUE

cost functions: -1 (maximize number of complexes covered)

2 (minimize number of selectors)

tolerance = 0 for both functions.

Let $e_0=l_1$, the elementary star ES_1 and the partial star P_1 contain:

ES_1 [x_1=0,2], [x_2=0..1], [x_3=5]

since multiplication by the entire space E (P_0) leaves each element unchanged. The trimmed P_1 is

P_1 [x_2=0..1], [x_3=5]

since each of these selectors cover at least 2 elements of L while the selector [x_1=0,2] covers only 1 element. The elementary star ES_2 is

ES_2 [x_2=0..2], [x_3=2]

since the references of the variable x_1 intersect in l_1 and l'_2. The resulting partial star P_2 is

P_2 $[x_2=0..1]$, $[x_2=0..1][x_3=2]$, $[x_2=0.2][x_3=5]$, $[x_3=2]$

which may be reduced by absorption to

P_2 $[x_2=0..1]$, $[x_2=0..2][x_3=5]$, $[x_3=2]$.

Since the third element of this partial star covers only one element of L, the trimed partial star leaves

P_2 $[x_2=0..1]$, $[x_2=0..2][x_3=5]$

Now considering l'_3, ES_3 is

ES_3 $[x_1=0,1]$, $[x_2=0..1]$, $[x_3=5]$

and P_3 after absorption becomes

P_3 $[x_2=0..1]$, $[x_2=0..2][x_3=5]$

The l_q selected from the star P_3 is

l_q $[x_2=0..2][x_3=5]$

since this complex covers all three complexes in L. Stripping reduces the complex to

$$l_q \quad [x_2=1..2][x_3=5]$$

If the set L' is ignored, a descriptive generalization of L can be formed

$$l_q \quad [x_1=0,1][x_2=1..2][x_3=5].$$

VITA

James B. Larson was born April 3, 1947 in Seattle Washington. He received a B.S. from the University of Washington in Chemistry and Mathematics in 1969 and an M. A. from the University of Washington in Mathematics in 1971. While at the University of Illinois, he was a teaching assistant in the Department of Computer Science for 4 years and a reasearch assistant under professor R.S. Michalski for 1 year.

The titles of his publications include: "A Multi-Step Formation of Variable-Valued Logic Hypotheses", "AQVAL/1-AQ7: User's Guide and Program Description" (with R. S. Michalski), "The DCS Information Storage and Retrieval System for Graduate Applicants and Students" (with W.J. Kubitz), and "Inductive Inference of VL Decision Rules" (with R. S. Michalski).

James Larson is a member of the Association for Computing Machinery and Sigma Xi.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-77-869	2.	3. Recipient's Accession No.
4. Title and Subtitle Inductive Inference in the Variable Valued Predicate Logic System VL_{21}: Methodology and Computer Implementation			5. Report Date May 1977 6.
7. Author(s) James B. Larson			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF MCS 74-03514
12. Sponsoring Organization Name and Address National Science Foundation Washington, DC			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

A formal methodology and computer program are presented for the transformation of a set of user supplied logical decision rules into a new, generalized set of decision rules which is near optimal according to a user supplied criterion. The VL$2 logic system (a multi-valued version of a first order predicate calculus) is used as the framework for defining and expressing decision rules and transformations on decision rules. The program INDUCE-1 which implements certain inductive inference rules using a graphical representation of VL$2 expressions is described and some examples of inductive problems solved by the program are given.

17. Key Words and Document Analysis. 17a. Descriptors

inductive learning
variable valued logic
inductive inference
decision rules
production systems

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages
	20. Security Class (This Page UNCLASSIFIED	22. Price